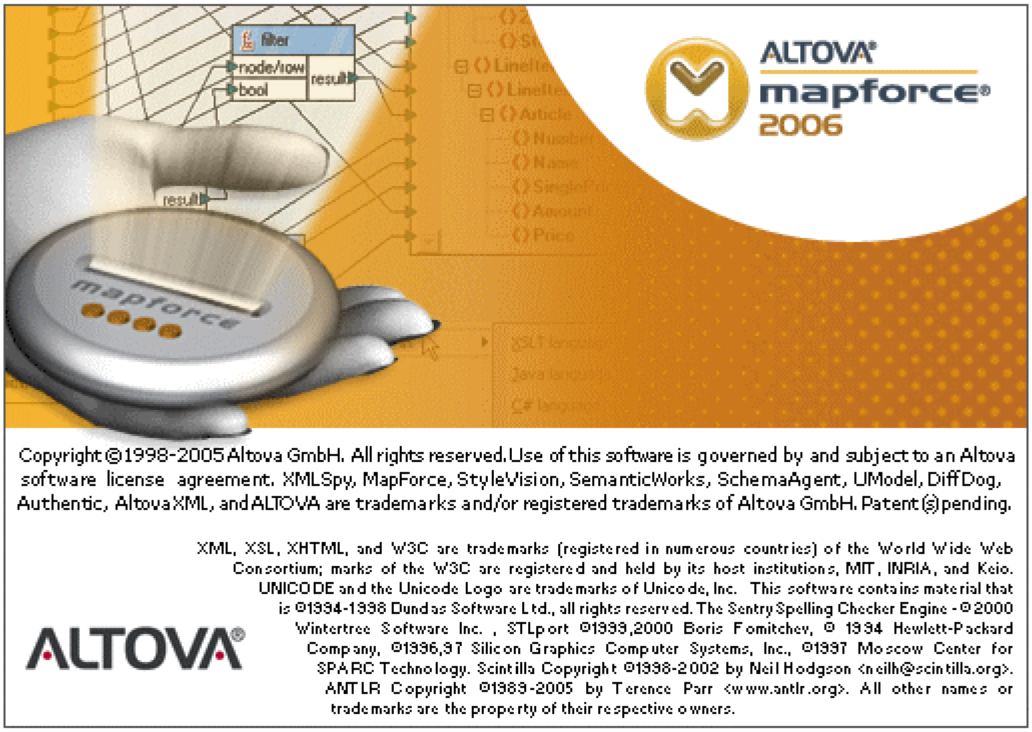
User and Reference Manual



Altova MapForce User Manual

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: 2006

Table of Contents

5 Copy-all connections 76

6 MapForce How To... 80

7 MapForce and Databases 102

23 The MapForce API 352

25 Appendices 472

Chapter 1

MapForce 2006

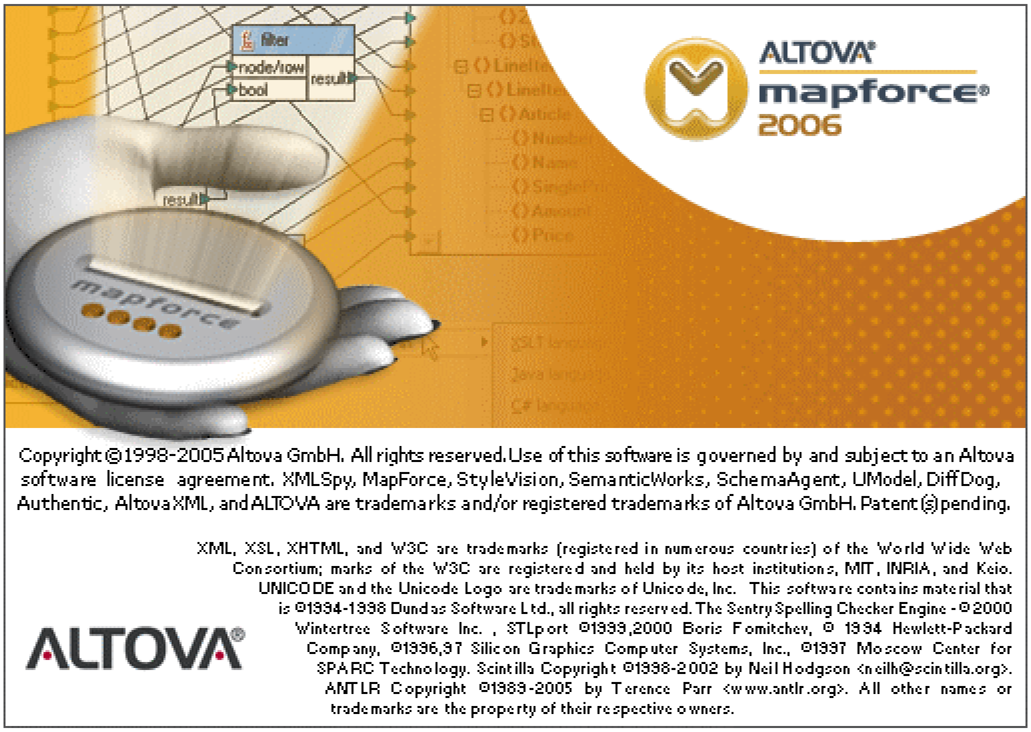
1 MapForce 2006

MapForce 2006 Professional Edition is a visual data mapping tool for advanced data integration projects.

MapForce can generate custom mapping code in XSLT 1.0 and 2.0, XQuery, Java, C#, and C++, and supports:

- Schema-to-Schema mapping
- Database-to-Schema/XML mapping
- XML-Schema-to-Database mapping
- Database-to-Database mapping
- Flat file mapping: CSV and Text files as source and target
- On-the-fly transformation and preview of database data, without code generation, or compilation
- Accessing MapForce user interface and functions through MapForce API (ActiveX control)
- Project management functions to group mappings
- Definition of custom XSLT 1.0 and 2.0 libraries
- Support for XPath 2.0 functions in XSLT 2.0 and XQuery
- Definition of user-defined functions/components, having complex in/outputs
- Advanced search and replace functions in transformation preview data
- XML-Schema substitution groups
- Support for source-driven / mixed content mapping
- MapForce plug-in for Eclipse
- MapForce for MS Visual Studio .NET

All transformations are available in one workspace where multiple sources and multiple targets can be mixed, and a rich and extensible function library provides support for any kind of data manipulation.



What is mapping?
Basically the contents of one component are mapped, or transformed, to another component. An XML, or text document, a database, can be mapped to a different target XML document, CSV text document, or database. The transformation is accomplished by an automatically generated XSLT 1.0 or 2.0 Stylesheet.

When creating an XSLT transformation, a **source schema** is mapped to a **target schema**. Thus elements/attributes in the source schema are "connected" to other elements/attributes in the target schema. As an XML document instance is associated to, and defined by, a schema file, you actually end up mapping two XML documents to each other.

Databases, can also be used as data sources, and map data to multiple XML Schemas, or other databases.

Chapter 2

MapForce overview

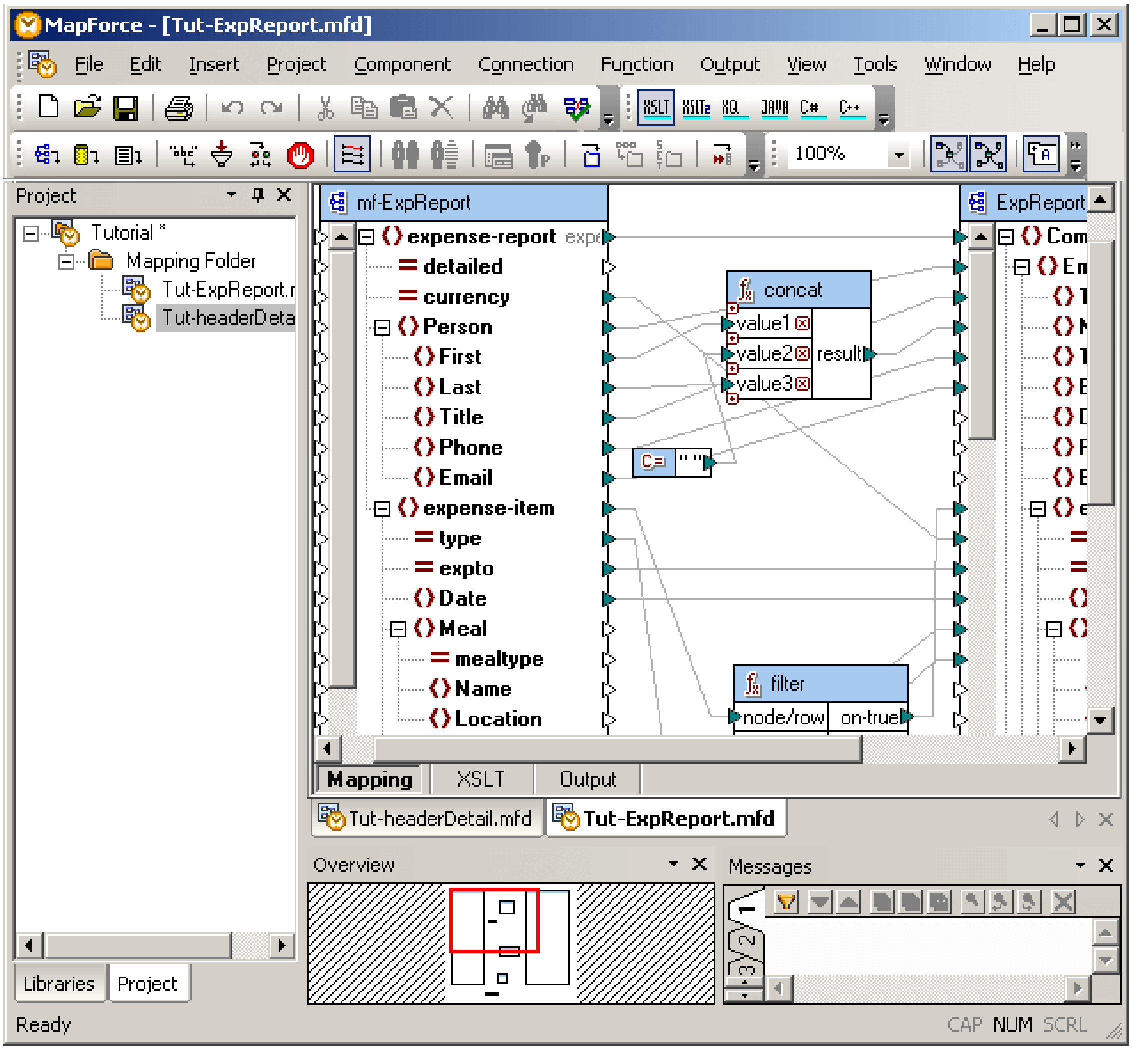
2 MapForce overview

MapForce has four main areas: the Library pane at left, the Mapping tab group at right, as well as the Overview and Messages panes below. The actual mapping process is achieved by manipulating the on-screen graphical elements in the mapping window.

- The **Libraries** pane displays language specific and user defined libraries, as well as the individual library functions. Functions can be directly dragged into the Mapping tab. The **Add Libraries....** button allows you to import external libraries into the tab group.
- The **Mapping** tab displays the graphical elements used to create the mapping (transformation) between the two schemas. The source schema is the "**mf-ExpReport**" component window displaying the source schema tree. The target schema is the " **ExpReport-Target**" window displaying the target schema tree. **Connectors** connect the input and output **icons** of each schema item. Schema **items** can be either elements or attributes.

  The **XSLT, XSLT2,** and **XQuery** tabs display a preview of the transformation depending on the specific language selected.

  The **Output** tab displays a preview of the transformed, or mapped data, in a text view.
- The **Overview** pane displays the mapping area as a red rectangle, which you can drag to navigate your Mapping.
- The **Messages** pane displays any validation warnings or error messages that might occur during the mapping process. Clicking a message in this pane, highlights it in the Mapping tab for you to correct.


MapForce - [Tut-ExpReport.mfd]
File
Edit
Insert
Project
Component
Connection
Function
Output
View
Tools
Window
Help
XSLT
XSLT2
XQ
JAVA
C#
C++
100%
Project
Tutorial *
Mapping Folder
Tut-ExpReport.r
Tut-headerDeta
Libraries
Project
mf-ExpReport
expense-report
expe
detailed
currency
Person
First
Last
Title
Phone
Email
expense-item
type
expto
Date
Meal
mealtype
Name
Location
concat
value1
value2
value3
result
C=
" "
filter
node/row
on-true
ExpReport
Com
En
Mapping
XSLT
Output
Tut-headerDetail.mfd
Tut-ExpReport.mfd
Overview
Messages
1
2
3
Ready
CAP
NUM
SCRL

2.1 Terminology

Library
A Library is a collection of functions visible in the Libraries window. There are several types of functions, core and language specific, as well as user-defined functions. Please see the section on functions for more details.

Component
In MapForce a component is a very generic "object". Almost all graphical elements you can insert/import or place in the Mapping tab, become components.

Components are recognizable by the small **triangles** they possess. These triangles (**input** and **output icons**) allow you to map data by creating a connection between them.

The following graphical elements are all components:

- All schema types: Source and target schemas
- All database types: Source and target databases
- All flat files: CSV and text files
-
- All function types: XSLT/XSLT2, XQuery, Java, C#, and C++ functions, as well as Constants, Filters and Conditions

Function
A function is basically an operation on data e.g. **Add**. Functions have input and/or output **parameters,** where each parameter has its own input/output icon. Functions are available in the Libraries window, and are logically grouped. Dragging a function into the Mapping window creates a function component. Please see the section functions and Libraries for more details.

Item
An item is the unit of data that can be mapped from schema to schema. An item can be either an **element**, an **attribute**, a database field.

Each **item** has an **input** and **output** icon which allows you to map data from one item to another. It is not mandatory that items be of the same type (element or attribute) when you create a mapping between them.

Input, Output icon
The small triangles visible on components are input and output icons. Clicking an icon and dragging, creates a **connector** which connects to another icon when you "drop" it there. The connector **represents a mapping** between the two sets of data the icons represent. Please see the section "Mapping between components" for more information.

Connector
The connector is the **line** that joins two icons. It represents the **mapping** between the two sets of data the icons represent. Please see the section "Mapping between components" for more information.

Several types of connector can be defined:

- Target Driven (Standard) connectors, see: "source-driven / mixed content vs. standard mapping"
- Copy-all connectors, please see "Copy-all connections"
- Source Driven (mixed content) connectors, see "source driven and mixed content mapping"

Constant
A constant is a component that supplies fixed data to an input icon. The data is entered into a

dialog box when creating the component. There is only one output icon on a constant function. You can select from the following types of data: Number, and All other (String).

Filter: Node/Row

A filter is a component that uses two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Boolean is true, then the value/content of the node/row parameter is forwarded to the **on-true** parameter.

The **on-false** output parameter, outputs the complement node set defined by the mapping, please see Multiple target schemas / documents for more information.

IF-Else Condition

A condition is a component which allows you to pass on different sets of data depending on the outcome of a preset condition. The component header displays the text **if-else**. Please see Condition, in the Reference section for an example.

- The first input parameter is a **bool**, which contains the data you are checking against.
- The **value-true** input parameter supplies the data to be passed on, as a result, if the condition is true.
- The **value-false** supplies the data to be passed on if the condition is false.
- The **result** parameter outputs the data supplied by the value-true/false input parameters.

2.2 MapForce components

When creating a mapping, single, or multiple **data sources**, can be mapped to multiple **target components.**

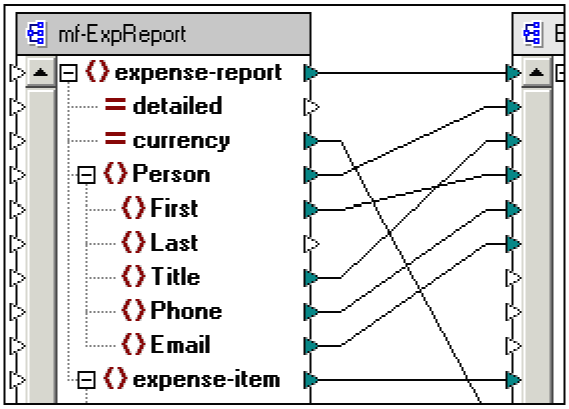
- Data sources can be: XML-Schemas/documents, CSV or text files, databases.
- Target components can be: XML-Schemas/documents, CSV or text files, as well as databases.

The mapping process allows the source data to be selectively transformed (or manipulated using functions) before it is output, or made available in the Output preview window.

A **data source** can have a **database,** CSV or text fileas its source. Once a data source has been imported/converted, it is used in exactly the same way as any of the other schema components in the Mapping tab.

To create a schema component (source):

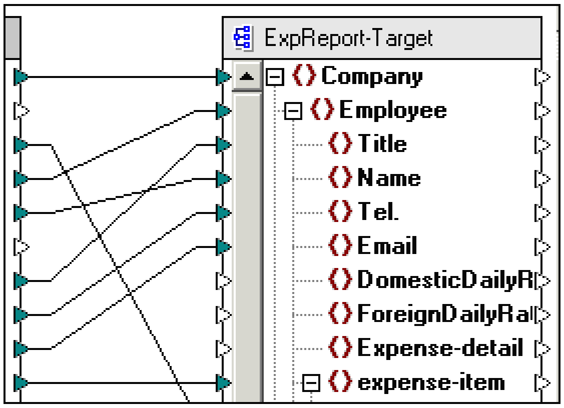
1. Click the **Insert | XML Schema/File** icon .
2. Select the schema file you want to use, from the "Open", dialog box.
   A further dialog box appears prompting you to select an **XML instance** file, if you intend to use this schema as a **data source** in this mapping.
3. Click **Yes** if this is the case, and select the **XML instance** file.
   The schema component now appears in the Mapping tab.



You can now connect the schema source output icons, with the target (or function) input icons, to create your mappings.

To create a schema component (target)

1. Click the **Insert | XML Schema/File** icon .
2. Select the schema file you want to use from the "Open", dialog box.
   Select **No** when you are prompted to supply an **XML instance** file.
3. Select the **Root element** of the schema you want to use (Company) and click OK.



The schema (snippet) with the root element appears as a schema component.

The **target schema** is the basis of the XML document you want to have **generated** by the transformation.

The target schema/document can, of course, differ dramatically from the source schema. This is where the mapping process comes in, you can map any item in the source schema/database to any other item (element/attribute), in the target schema. The source data then appears at the position defined by your mapping, in the target document.

You can also define multiple output schemas. MapForce then generates XSLT, XQuery, or program code for each target schema. You can then selectively preview the different output schemas in the Output preview window, please see the section "XSLT and Output previews" for more information.

Please note:

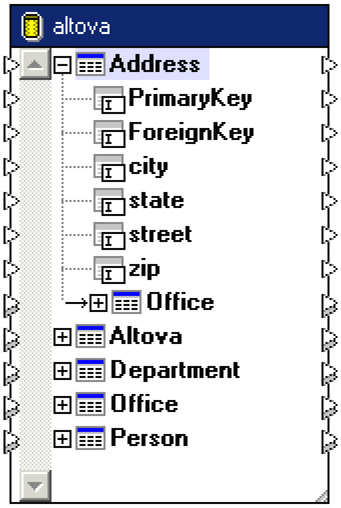
It is not necessary to associate an XML Instance document to a **target schema**. If you do so, then the XML instance document is ignored and does not affect the transformation in any way.

Clicking the root element of a schema and hitting the ***** key on the numeric keypad, expands all the schema items!

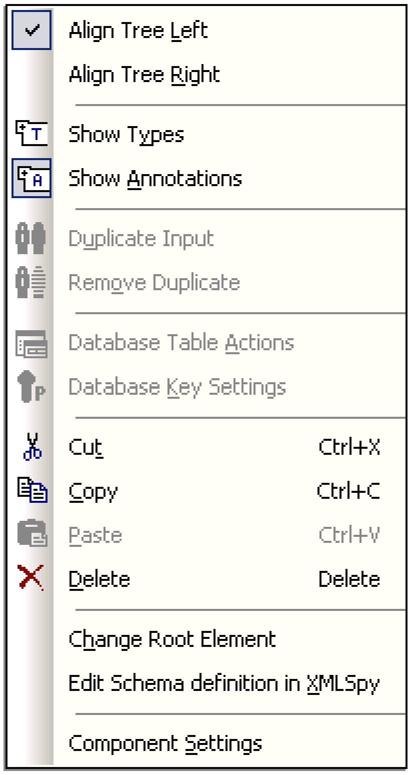
To create a Database component (source/target):

The database structure is the basis of the component and is displayed it in a tree view.

1. Click the **Insert Database** icon .
2. Select the source **database type** by clicking on one of the radio buttons (e.g. Microsoft Access), and click Next.
3. Click **Browse** to select navigate and select an Access database, (e.g. **Tutorial\altova.mdb**) and then click Next.
4. Select the database tables you want to import, or have access to (Select All).
5. Click the **Insert Now** button at the bottom of the dialog box.
   The database component now appears in the Mapping window.



Schema component **context menu**
Right clicking a schema component in the Mapping window opens the context menu.



Align tree left
Aligns all the items along the left hand window border. This display is useful when creating mappings from the source schema.

Align tree right
Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Show types

Displays the schema data type of all the schema items.

Show annotations
Shows schema annotations.

Duplicate input
Inserts a copy/clone of the selected item, allowing you to map multiple input data to this item. Duplicate items do not have output icons, you cannot use them as data sources. Please see the Duplicating input items section in the tutorial for an example of this.

Remove duplicate
Removes a previously defined duplicate item. Please see the Duplicating input items section in the tutorial for more information.

Database Table actions
Allows you to define the table actions to be performed on the specific target database table. Table actions are: Insert, Update, and Delete, please see Mapping data to databases for more information.

Database Key settings
Allows you to define the Key settings of database fields, please see Database Key settings for more information.

Cut/Copy/Paste/Delete
The standard MS Windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window. All connectors will be retained except for those which would have to be replaced.

Change Root element
Allows you to change the root element of the XML instance document. Useful in the target schema window, as this limits or preselects the schema data.

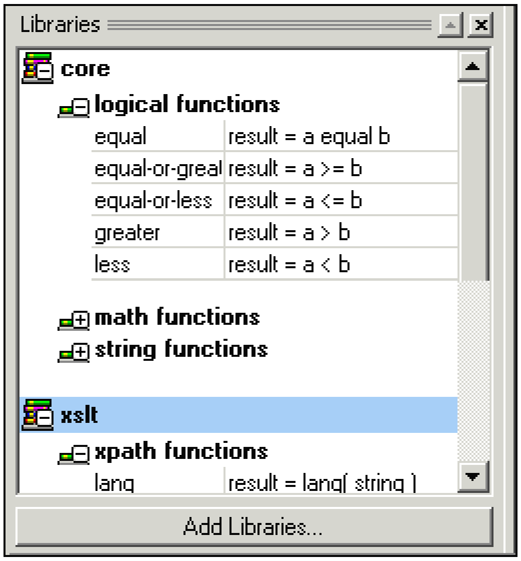
Edit Schema definition in XMLSpy.
Starts XMLSpy and opens the schema file, ready for you to edit.

Component Settings
Opens the Component Settings dialog box.

Allows you to select the input and/or output XML Instance, as well as define database specific settings for code generation. Please see **Component Settings** for more information on these settings.

2.3 Functions and libraries

The **Libraries** pane displays the available libraries for the currently selected programming language, as well as the individual **functions** of each library. Functions can be directly dragged into the **Mapping** tab. Once you do this, they become function components.



equal	result = a equal b
equal-or-great	result = a >= b
equal-or-less	result = a <= b
greater	result = a > b
less	result = a < b
lang	result = lang(string)

The standard **core, lang, xpath2** and **xslt** libraries are always loaded when you start MapForce , and do not need to be added by the user. The **Core** library is a collection of functions that can be used to produce all types of output: XSLT, XQuery, Java, C#, C++,. The other libraries (XSLT, XSLT2, XPath2, Lang etc.) contain functions associated with each separate type of output.

Please note:
The XPath 2.0 library and its functions, are common to both XSLT 2.0 and XQuery languages

Selecting:
XSLT, enables the core and XSLT functions (XPath 1.0 and XSLT 1.0 functions).

XSLT2, enables the core, XPath 2.0, and XSLT 2.0 functions.

XQ(uery), enables the core and XPath 2.0 functions.

XPath 2.0 restrictions:
Several XPath 2.0 functions dealing with sequences are currently not available.

To use a function in Mapping window:

1. First select the **programming language** you intend to generate code for, by clicking one of the output icons in the title bar: XSLT/XSLT2 XQ, Java, C#, or C++.
   The functions associated with that language are now visible in the Libraries window. The expand and contract icons show, or hide the functions of that library.
2. Click the **function name** and drag it into the Mapping window.
3. Use drag and drop to connect the input and output parameters to the various icons.

   Note that placing the mouse pointer over the "result = xxx" expression in the library pane, displays a ToolTip describing the function in greater detail.

Function tooltips:
Explanatory text (visible in the libraries pane) on individual functions, can now be toggled on/off

by clicking the "Show tips" icon INFO in the title bar. Placing the mouse pointer over a function header, displays the information on that function.

To add new function libraries:

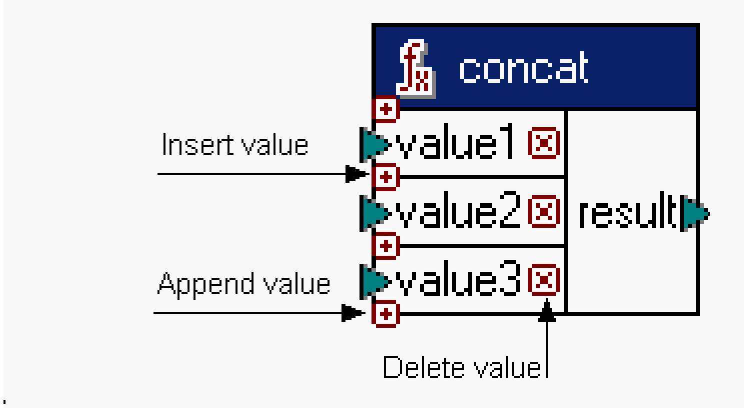
MapForce allows you to create and integrate your own function libraries please see the sections: "Adding custom function libraries, "Adding custom XSLT 1.0 functions" "Adding custom XSLT 2.0 functions" and "User-defined functions" for more information.

Please note:
custom functions/libraries can be defined for Java, C#, and C++, as well as for XSLT.

Extendable functions

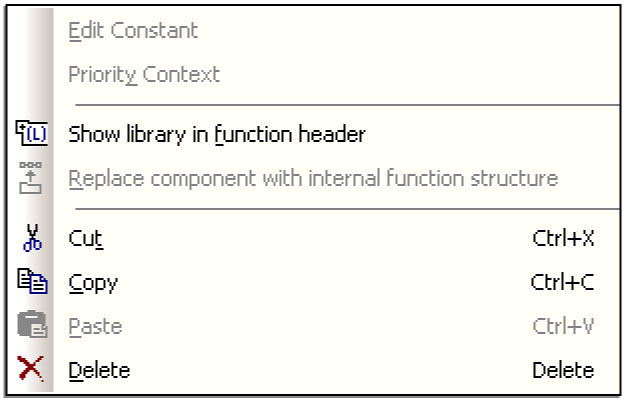
Several functions available in the function libraries are extendable: for e.g. the concat and "logical-and" functions. The parameters of these types of function can be inserted/appended and deleted at will. Clicking the "plus" icon inserts or appends the same type of parameter, while clicking the check mark deletes the parameter.

Please note: "dropping" a connector on the "plus" symbol, automatically inserts/appends the parameter and connects it.



Function context menu:

Right clicking a function in the Mapping window, opens the context window.


Edit Constant
Priority Context
Show library in function header
Replace component with internal function structure
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Delete

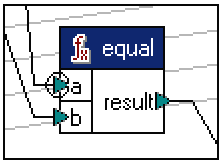
Edit Constant

Allows you to change the entry currently defined in the Constant component. A Constant is added by clicking the **Insert Constant** icon "abc".

Priority Context

When applying a function to different items in a schema or database, MapForce needs to know what the context node will be. All other items are then processed relative to this one.

This is achieved by designating the item (or node) as the priority context. A circle appears around the icon so designated. Please see Priority Context in the Reference section, for an example.



Show library in function header
Displays the library name in the function component.

Replace component with internal function structure
Replaces the user-defined component/function with its constituent parts.

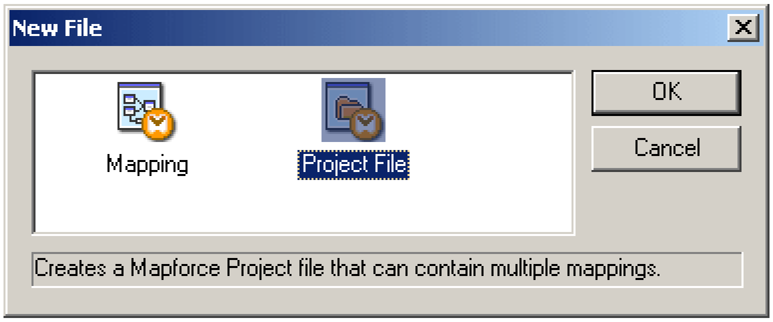
Cut/Copy/Paste/Delete
The standard MS Windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window. All connectors will be retained except for those which would have to be replaced.

2.4 Projects

MapForce supports the Multiple Document Interface, and allows you to group your mappings into mapping projects. Project files have a ***.mfp** extension.

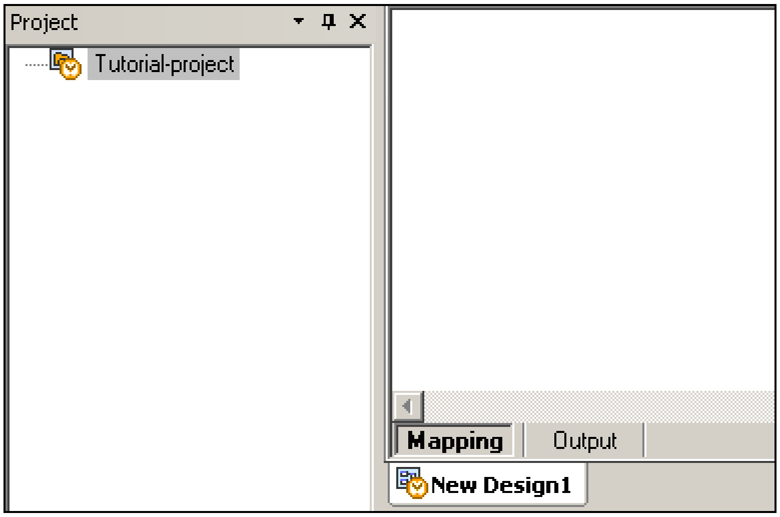
To create a project:

1. Select **File | New** and double click the Project File icon.



Creates a Mapforce Project file that can contain multiple mappings.

2. Enter the project name in the **Save Project As** dialog box, and click Save to continue. A project folder is added to the Project tab.



3. Select **File | New** and double click the "Mapping" icon. This opens a new mapping file, "New Design1", in the Design pane.

To add mappings to a project:

1. Select **Project | Add active file to project**. This adds the currently active file to the project. The mapping name now appears below the project name in the project tab.

- Selecting the option **Project | Add files to project**, allows you to add files that are not currently opened in MapForce.

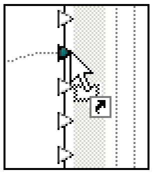
To remove a mapping from a project:

1. Right click the mapping icon below the project folder,
2. Select Remove mapping from the pop-up menu.

2.5 Mapping between components

A **connector** visualizes the **mapping** between the two sets of data and allows the **source** data (value) to appear, or be transformed, into the target schema/document or database.

- **Components** and **functions** have small "connection" triangles called: **input** or **output** icons. These **icons** are positioned to the left and/or right of all "mappable" **items**.
- Clicking an icon and dragging, creates the **mapping connector**. You can now drop it on another icon. A link icon appears next to the text cursor when the drop action is allowed.

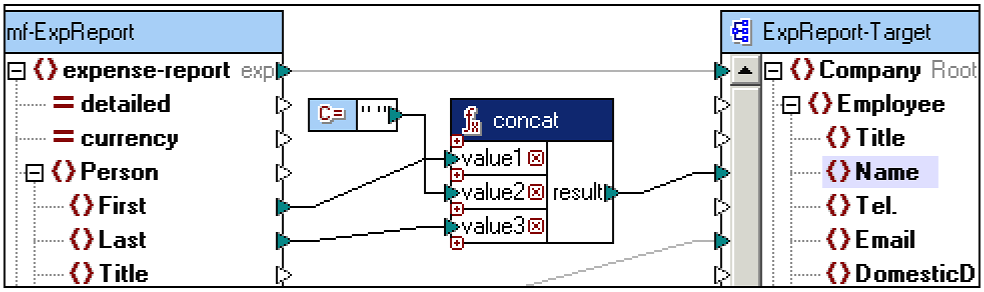


- Clicking an **item name** (element/attribute) automatically selects the correct **icon** for the dragging action.
- An **input icon** can only have one connector.
  If you try and connect a second connector to it, a prompt appears asking if you want to replace or **duplicate** the input icon.
- An **output icon** can have several connectors, each to a different input icon.
- Placing the mouse pointer over the straight section of a connector (close to the input/output icon) highlights it. You can now reposition the connector by dragging it elsewhere.

Number of connectors
Input and output icons appear on most components, there is not, however, a one to one relationship between their numbers.

- Each **schema item** (element/attribute) has an input and output icon.
- **Database** items have input and output icons.
- Duplicated items only have input icons. This allows you to map multiple inputs to them. Please see Duplicating Input items for more information.
- **Functions** can have any number of input and output icons, **one** for each **parameter**. E.g. the Add Function has two input icons, and one output icon.
- **Special** components, can have any number of icons, e.g. the Constant component only has an output icon.

This example shows how you can use the **concat** function to combine the First and Last names and place the result in the Title element. The constant component, supplies the space character between the two names.



2.5.1 Connector properties

Connectors and their properties:

- Clicking a connector highlights it in red.
- Hitting the **Del** key, while highlighted, deletes it immediately.
- Right clicking a connector, opens the connector context menu.
- Double clicking a connector, opens the Connection Settings dialog box.

Viewing connectors

MapForce allows you to selectively view the connectors in the mapping window.

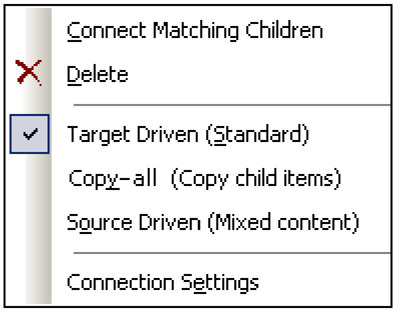
Show selected component connectors
Switches between showing:

- all mapping connectors, or
- those connectors relating to the currently selected component.

Show connectors from source to target
Switches between showing:

- connectors that are **directly** connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

Connector context menu:


Connect Matching Children
Delete
Target Driven (Standard)
Copy-all (Copy child items)
Source Driven (Mixed content)
Connection Settings

Connect matching children
Opens the "Connect Matching Children" dialog box, allowing you to change the connection settings and connect the items when confirming with OK.

Delete
Deletes the selected connector.

Target Driven (Standard)
Changes the connector type to Standard mapping, please see: "Source-driven / mixed content vs. standard mapping" for more information.

Copy-all
Changes the connector type to "Copy-all" and connects all child items of the same name in a graphically optimized fashion, please see "Copy-all connections" for more information.

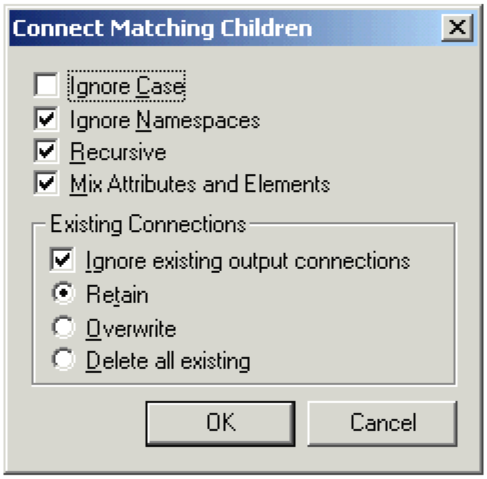
Source Driven (mixed content)
Changes the connector type to source-driven / mixed content, please see: "Source driven and

mixed content mapping" for more information.

Connection settings:
Opens the Connections Settings dialog, in which you can define the specific mixed content settings as well as the connector annotation settings, please see the Connection section in the Reference section.

Connect matching Children dialog box
This command allows you to create multiple connectors between items of the **same name** in both the source and target components.

1. Connect two (parent) items that share identically named **child items** in both components.
2. Right click the connector and select the **Connect matching child elements** option.



Connect Matching Children

☐ Ignore Case
☑ Ignore Namespaces
☑ Recursive
☑ Mix Attributes and Elements

Existing Connections
☑ Ignore existing output connections
● Retain
○ Overwrite
○ Delete all existing

3. Select the required options discussed in the text below, and click OK to create the mappings.

   Mappings are created for all the child items that have identical names and adhere to the settings defined in the dialog box.

Please note:
The settings you define here are retained, and are applied when connecting two items, if the "**Auto connect child items**" icon in the title bar is active. Clicking the icon, switches between an active and deactive state.

Ignore Case:
Ignores the case of the child item names.

Ignore Namespaces:
Ignores the namespaces of the child items.

Recursive:
Having created the first set of connectors, the grandchild items are then checked for identical names. If some exist, then connectors are also created for them. The child elements of these items are now checked, and so on.

Mix Attributes and Elements:
Allows the creation of connectors between items of the same name, even if they are of different types e.g. two "Name" items exist, but one is an element, the other an attribute. If set active, a connector is created between these items.

Existing connections:

Ignore existing output connections:
Creates **additional** connectors to other components, even if the currently existing output icons already have connectors.

Retain
Retains existing connectors.

Overwrite:
Recreates connectors, according to the settings defined. Existing connectors are scrapped.

Delete all existing:
Deletes all existing connectors, before creating new ones.

2.6 Validating mappings and mapping output

Validating a Mapping validates:

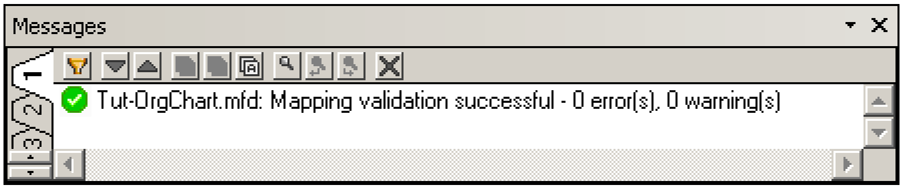
- that all mappings (connectors) are valid
- please note, that the current release supports mixed content mapping.

Connectors and validation
It is not mandatory for functions or components to be mapped. The Mapping tab is a work area where you can place any available components. XSLT 1.0, XSLT 2 XQuery, Java, C#, or C++ code is only generated for those components for which valid connections exist.

To validate your mapping:

- Click the Validate Mapping icon , or select the menu item **File | Validate Mapping.**
- Click one of the preview tabs, (XSLT, XSLT 2.0, or Output), or
- Select the menu option **File | Generate XSLT/XSLT2, Generate XQuery, Java, C#, or C++ code**

A validation message appears in the Messages window.



Tut-OrgChart.mfd: Mapping validation successful - 0 error(s), 0 warning(s)

Note that you can use multiple message tabs if you project contains many separate mapping files. Click one of the numbered tabs in the Messages window, and click the preview tab for a different mapping in your project. The validation message now appears in the tab that you selected. The original message in tab 1, is retained however.

Use the different icons of the Messages tab to:

- Filter the message types, errors or warnings
- Scroll through the entries
- Copy message text to the clipboard
- Find a specific string in a message
- Clear the message window.

Validation messages:

- Validation successful - X Error(s), Y Warning(s).

Warnings, alert you to something, while still enabling the mapping process and preview of the transformation result to continue. It is therefore possible for a mapping to have 0 errors and Y warnings.

Errors, halt the transformation process and deliver an error message. An XSLT, XQuery, or Output preview is not possible when an error of this type exists. Clicking a validation message in the Messages window, highlights the offending component icon in the Mapping window.

Component connections and validation results:

Free standing components

- Do not generate any type of error or warning message.

Partially connected components can generate two types of warning:

- If a function component **input icon** is unconnected, an error message is generated and the transformation is halted.
- If the function **output icon** is unconnected, then a warning is generated and the transformation process continues. The offending component and its data are ignored, and are not mapped to the target document.

Validating mapped OUTPUT:
Clicking the Output tab uses the MapForce, XSLT 1.0/2.0 or XQuery engine, to transform the data and produce a result in a Text view. If the data is mapped to an XML Schema, then the resulting XML document can be validated against the underlying schema.

- Click the Validate button ✓ to validate the document against the schema. A "Output XML document is valid" message, or a message detailing any errors appears.

2.7 XSLT, Output tab - generating XSLT or program code

The XSLT, XSLT2, XQuery, and Output tabs of the Mapping tab group, supply a **preview** of:

- the generated XSLT, or XQuery code and
- the resulting transformation produced by the MapForce engine.

Please note:

The result generated by the MapForce engine, is an on-the-fly transformation of database, Text, without you having to generate, or compile program code! We would recommend that you use this option until you are satisfied with the results, and then generate program code once you are done. The generated program code will have a much faster execution speed.

To save the generated XSLT code:

1. Select the menu option **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Browse for the folder where you want to save the XSLT file.
3. A message appears when the generation was successful.
4. Navigate to the previously defined folder, where you will find the generated XSLT file.

To save the XML, or output data from the Output tab:

1. Click the Output tab to preview the mapping result.
2. Click the "**Save generated output**" icon 💾, and specify where you want the result to be saved.

If the target is an **XML/Schema** file:

- The Save generated output icon is active. Click it to save the output.

If the target is a **Database**:

- The Run SQL-script icon is active. Click it to update, insert, or delete the database data.

To transform an XML/Schema file using the generated XSLT:

1. Open the XML file in the editor of your choice (**XMLSpy** for example).
2. Assign the XSLT file to the XML file (**XSL/XQuery | Assign XSL**).
3. Start the transformation process (**XSL/XQuery | XSL Transformation**).
   The transformed XML document appears in your editor.

To generate program code:

1. Select the specific menu option: **File | Generate code in | XSLT/XSLT2, XQuery, Java, C#, C++**)
2. Browse for the folder where you want to save the program files.
3. A message appears when the code generation was successful.
4. Compile and execute the code using your specific compiler.

Please note:

A **JBuilder** project file and **Ant** build scripts are generated by MapForce to aid in compiling the Java code, see the section on JDBC driver setup as well as the code generator section for more information.

To search for specific data in the Output tab:

- Select the menu option **Edit | Find**, or hit the **CTRL+F** keyboard keys.
  The Find dialog box allows you to specify the search options in great detail, and also supports regular expressions.

Chapter 3

MapForce tutorial

3 MapForce tutorial

Tutorial example:
In the tutorial, a simple employee travel expense report will be mapped to a more complex company report.

Each employee fills in the fields of the personal report. This report is mapped to the company report and routed to the Administration department. Extra data now has to be entered in conjunction with the employee, the result being a standardized company expense report.

Further formatting, cost summation, and conditional viewing options of the expense report, are made possible by having the target XML document associated with StyleVision Power Stylesheet designed in StyleVision.

Aim of the tutorial:

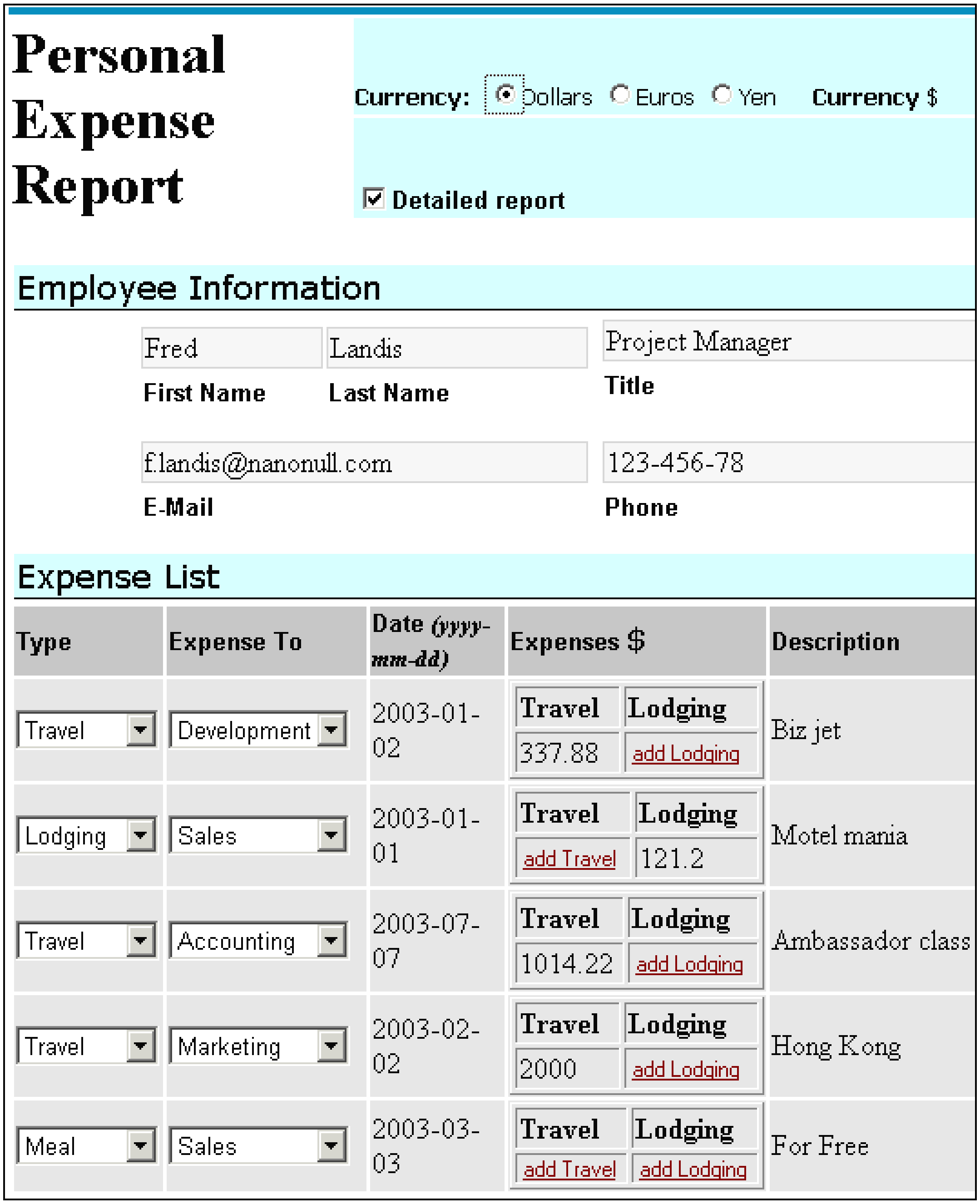
- To transform the **personal expense report** to a company expense travel report
- **Selectively filter** the source data and only let the travel expense records through
- Generate an XSLT transformation file
- Transform the personal expense report to the company expense report using the generated XSLT file
- Assign an StyleVision Power Stylesheet to the resulting XML file, enabling you to view and edit the resulting file in the Authentic View

The tutorial makes use of the following components:

- source and (multiple) target schemas
- an MS Access database as the data source
- several functions including: concat, filter, equal and constants

Files used in the tutorial:
All the files used in this tutorial are available in the **...\MapForceExamples\Tutorial** folder. The XSLT and transformed XML files are also supplied.

Tutorial files:	**Personal expense report**
Tut-ExpReport.mfd	The expense report mapping (single target)
Tut-ExpReport-multi.mfd	The multi-schema target expense report mapping
PersonDB.mfd	The employee mapping, using an MS Access DB as the data source
mf-ExpReport.xml	Personal expense report XML instance document
mf-ExpReport.xsd	Associated schema file
mf-ExpReport.sps	StyleVision Power Stylesheet used to view the personal expense report in Authentic View of XMLSpy, or Authentic Desktop.
	Company expense report
ExpReport-Target.xml	Company expense report XML instance document
ExpReport-Target.xsd	Associated schema file
ExpReport-Target.sps	StyleVision Power Stylesheet used to view the Company expense report in Authentic View of XMLSpy, or Authentic Desktop.



Personal Expense Report

Currency: ● Dollars ○ Euros ○ Yen Currency $

☑ Detailed report

Employee Information

Fred	Landis	Project Manager
First Name	**Last Name**	**Title**

f.landis@nanonull.com	123-456-78
E-Mail	**Phone**

Expense List

Type	Expense To	Date *(yyyy-mm-dd)*	Expenses $	Description
Travel	Development	2003-01-02	Travel Lodging 337.88 add Lodging	Biz jet
Lodging	Sales	2003-01-01	Travel Lodging add Travel 121.2	Motel mania
Travel	Accounting	2003-07-07	Travel Lodging 1014.22 add Lodging	Ambassador class
Travel	Marketing	2003-02-02	Travel Lodging 2000 add Lodging	Hong Kong
Meal	Sales	2003-03-03	Travel Lodging add Travel add Lodging	For Free

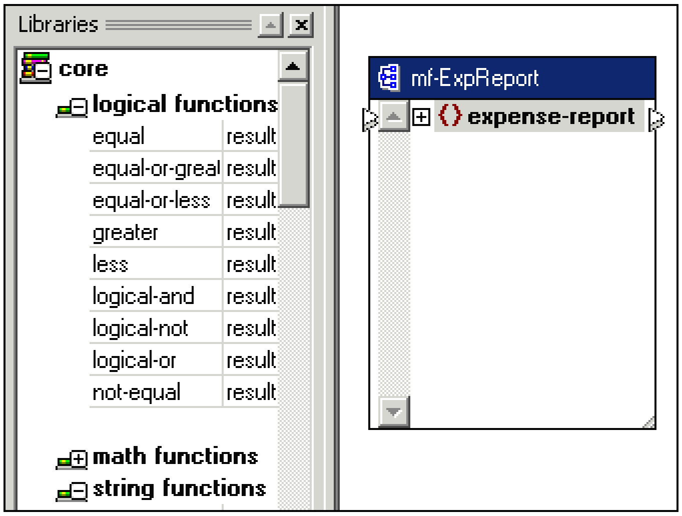
3.1 Setting up the mapping environment

This section deals with defining the source and target schemas we want to use for the mapping.

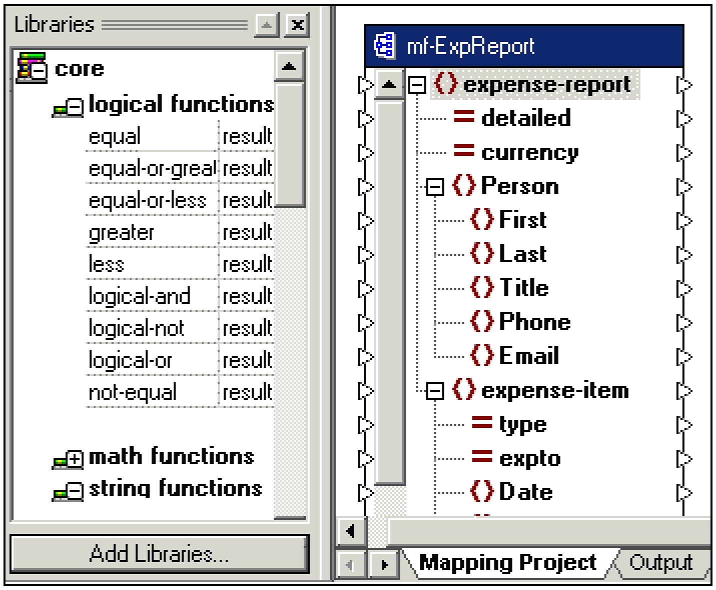
- Start MapForce.

Creating the source schema component:

1. Click the **Insert XML Schema/File** icon.
2. Select the **mf-ExpReport.xsd** file from the Open dialog box.
   You are now prompted for a sample XML file to provide the data for the preview tab.
3. Click Yes, and select the **mf-ExpReport.xml** file.
   The source schema component now appears in the Mapping tab.

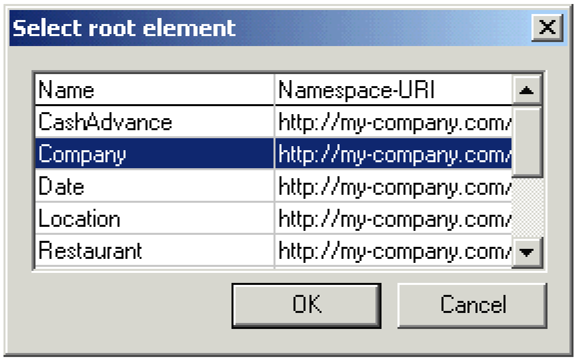


4. Click the **expense-report** entry and hit the * key, on the numeric keypad, to view all the items.
5. Click the expand icon at the lower right of the component window, and resize the window.



Creating the target schema component:

1. Click the **Insert XML Schema/File** icon.
2. Select the **ExpReport-Target.xsd** file from the Open dialog box.
   You are now prompted for a sample XML file for this schema.
3. Click No, and select **Company** as the root element of the target document.

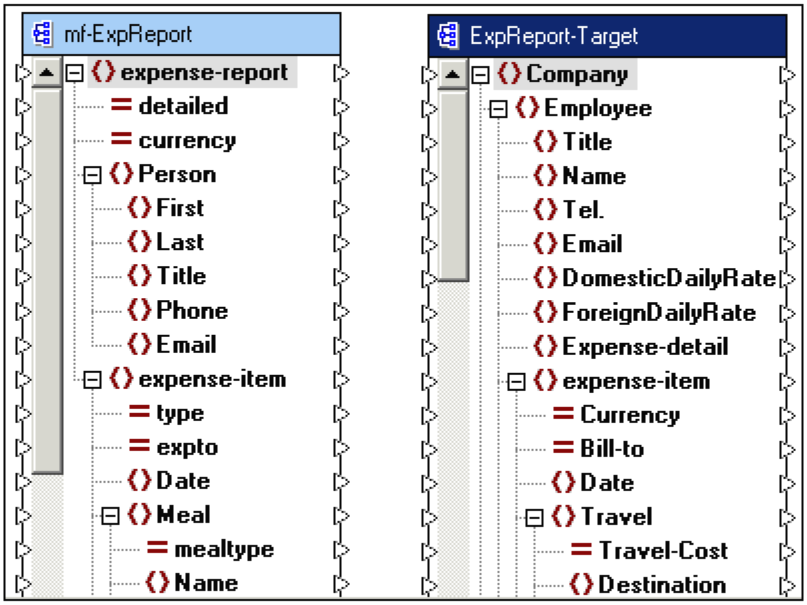


Select root element

Name	Namespace-URI
CashAdvance	http://my-company.com/
Company	http://my-company.com/
Date	http://my-company.com/
Location	http://my-company.com/
Restaurant	http://my-company.com/

The target schema component now appears in the mapping tab.

4. Click the **Company** entry and hit the ***** key on the numeric keypad to view all the items.
5. Click the expand window icon and resize the window.

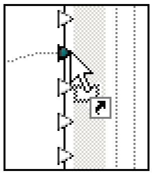


We are now ready to start mapping schema items from the source to the target schema.

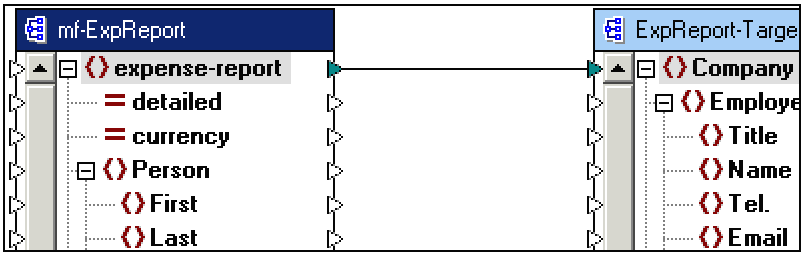
3.2 Mapping schema items

This section deals with defining the mappings between the source and target schema items.

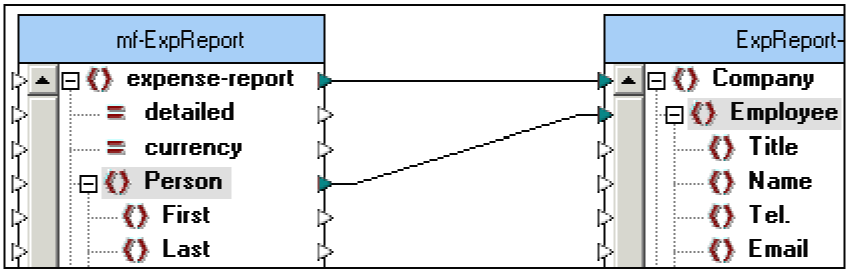
1. Click the **expense-report** item in the source schema and drag.
   A connector line is automatically created from the output icon and is linked to the mouse pointer which has now changed shape.
2. Move the mouse pointer near to the **Company** item in the target schema, and "drop" the connector the moment the mouse pointer changes back to the arrow shape. A small link icon appears below the mouse pointer, and the input icon is highlighted when the drop action will be successful.



A connector has now been placed between the source and target schemas. A mapping has now been created from the schema source to the target document.



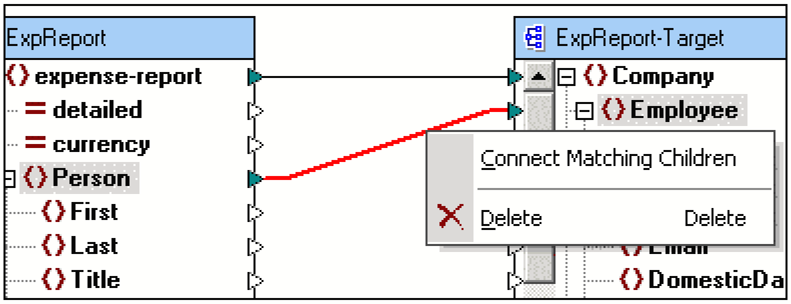
3. Use the above method to create a mapping between the Person and Employee items.



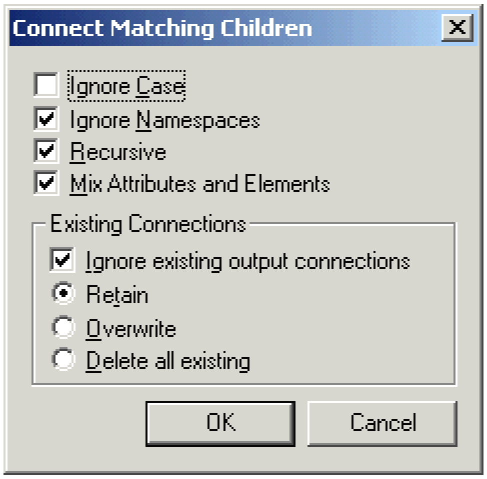
Auto-mapping

MapForce allows you to automatically connect child elements of the same name in both schemas. For more information please see the section on Connector properties.

1. Right click the "Person" connector and select "Connect matching children" from the pop-up menu.
   If the child items are automatically connected, auto connect child items is active.



This opens the Connect Matching Children dialog box.

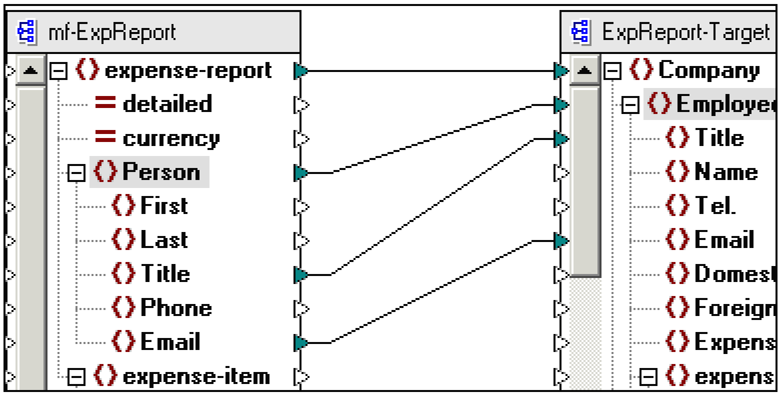


Connect Matching Children

☐ Ignore Case
☑ Ignore Namespaces
☑ Recursive
☑ Mix Attributes and Elements

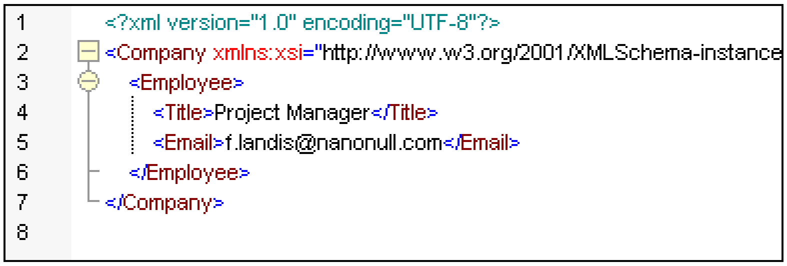
Existing Connections
☑ Ignore existing output connections
● Retain
○ Overwrite
○ Delete all existing

2. Activate all four check boxes, and click OK.



Mappings have been automatically created for the **Title** and **Email** items of both schemas.

3. Click the Output tab to see if there is a result.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
	<Employee>
		<Title>Project Manager</Title>
		<Email>f.landis@nanonull.com</Email>
	</Employee>
</Company>

```

You will notice that the Title and Email fields contain data originating from the XML Instance document.

4. Click the Mapping tab to continue mapping.

Please note:

The settings you select in the Connect Matching Children dialog box, are retained until you change them. These settings can be applied to a connection by either: using the context menu, or by clicking the Auto connect child items icon to activate, or deactivate this option.

3.3 Using functions to map data

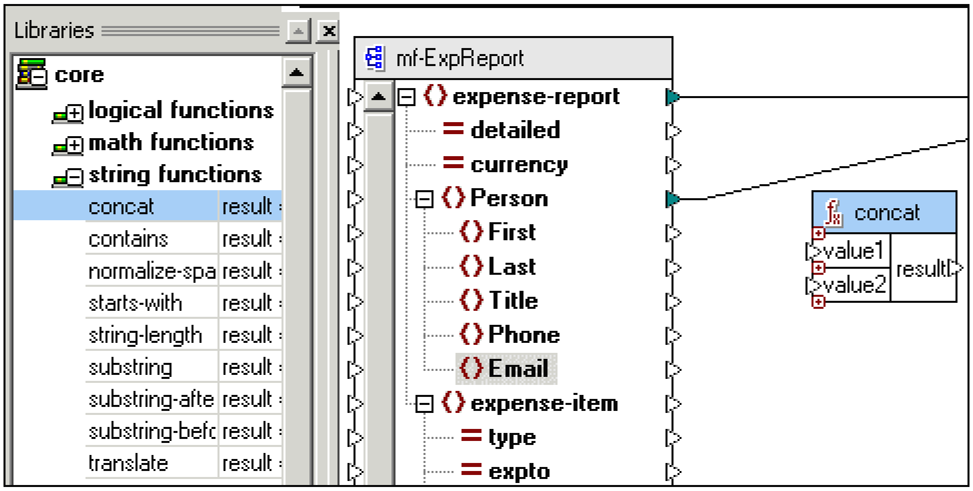
The aim of this section is to combine two sets of data from the source schema, and place the result in a single item in the target document. Please note, that some of the previously defined mappings are not shown in the following screen shots for the sake of clarity.

This will be done by:

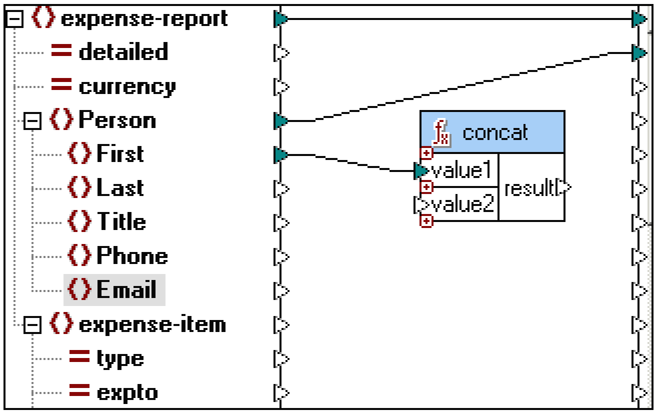
- Using the **Concat** string function to combine the **First** and **Last** elements of the source schema
- Using a **Constant** function to supply the space character needed to separate both items
- Placing the result of this process into the **Name** item of the target schema.

Using functions to combine items:

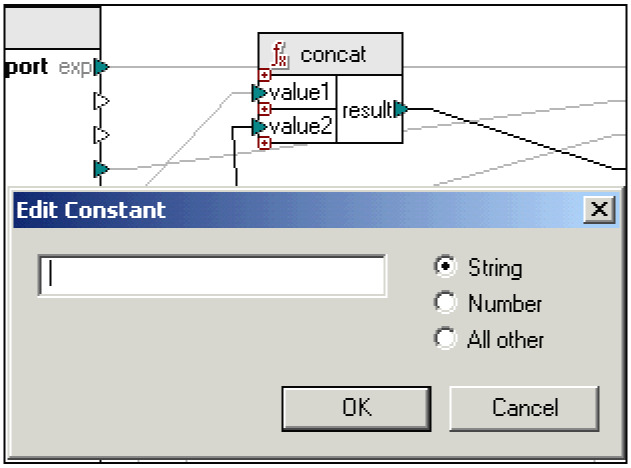
1. Click the **concat** entry of the string functions group, in the Core library, and drag it into the Mapping tab



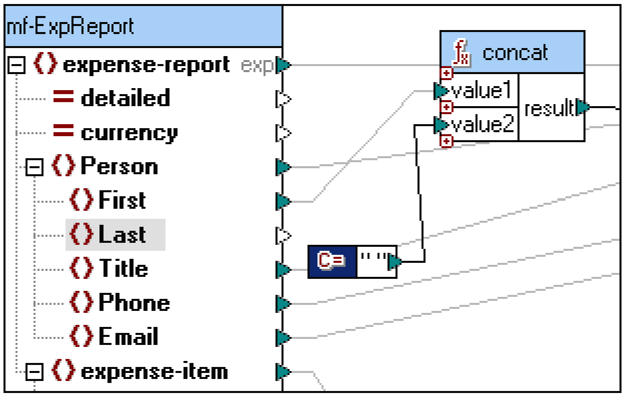
2. Create a connection between item **First** and **value1** of the concat component.



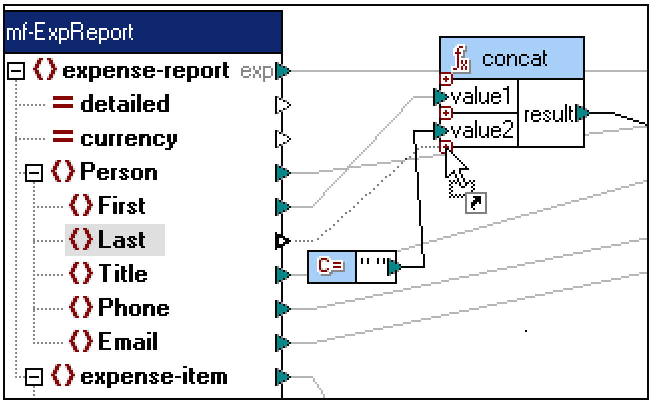
3. Click the **Insert Constant** icon "abc" in the icon bar, to insert a constant component.



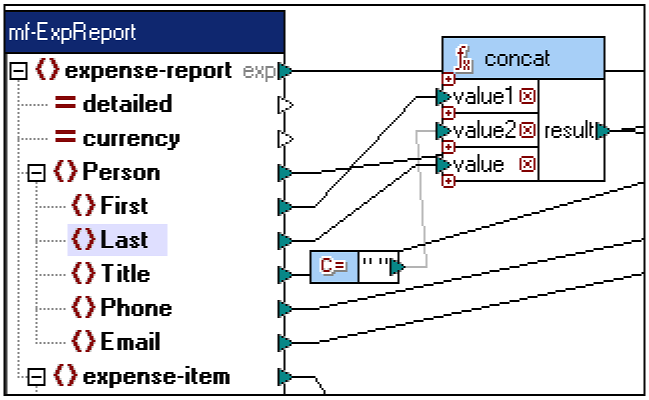
4. Enter a space character in the text box and click OK.
 The constant component is now in the working area. Its contents are displayed next to the output icon.
5. Create a connection between the **constant** component and **value2** of the concat component.



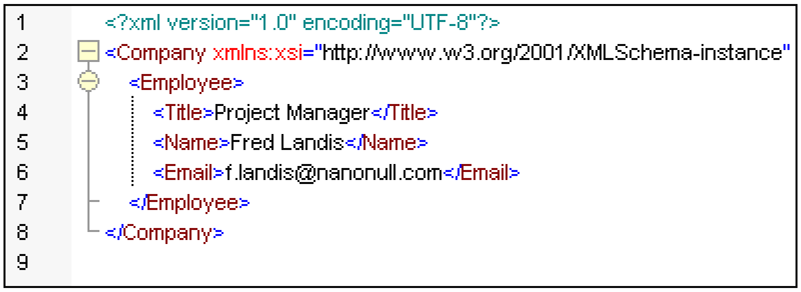
6. Click the item **Last** and drop the connector on the "**+**" icon of the concat function, just below **value2**. The text cursor changes to show when you can drop the connector.



This automatically enlarges the concat function by one more item (value), to which the Last item is connected.



7. Connect the **result** icon of the concat component, to the **Name** item in the target schema.
8. Click the **Output** tab to see the result of the current mapping.

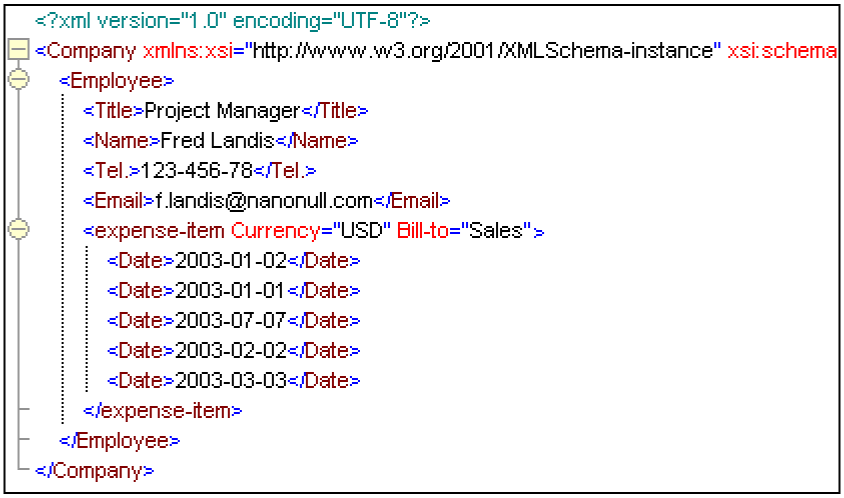


```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Employee>
        <Title>Project Manager</Title>
        <Name>Fred Landis</Name>
        <Email>f.landis@nanonull.com</Email>
    </Employee>
</Company>

```

You will see that the Person name "Fred Landis" is now contained between the **Name** tags. The first and last name have been separated by a space character as well.

Mapping the rest of the personal data:

1. Create mappings between the following items:
   - currency to Currency
   - Phone to Tel.
   - expto to Bill-to
   - Date to Date
2. Click the Output tab to see the result.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schema
    <Employee>
        <Title>Project Manager</Title>
        <Name>Fred Landis</Name>
        <Tel.>123-456-78</Tel.>
        <Email>f.landis@nanonull.com</Email>
        <expense-item Currency="USD" Bill-to="Sales">
            <Date>2003-01-02</Date>
            <Date>2003-01-01</Date>
            <Date>2003-07-07</Date>
            <Date>2003-02-02</Date>
            <Date>2003-03-03</Date>
        </expense-item>
    </Employee>
</Company>
```

There are currently five items originating from the assigned XML instance file.

Please note:
Functions can be grouped into user-defined functions/components to maximize screen space. Please see the section on "User-defined functions/components" for an example on how to combine the concat and constant functions into a single user-defined function/component.

3.4 Filtering data

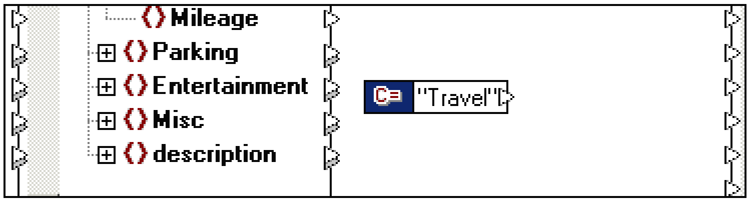
The aim of this section is to filter out the Lodging and Meal expenses, and only pass on the Travel expenses to the target schema/document.

This will be done by:

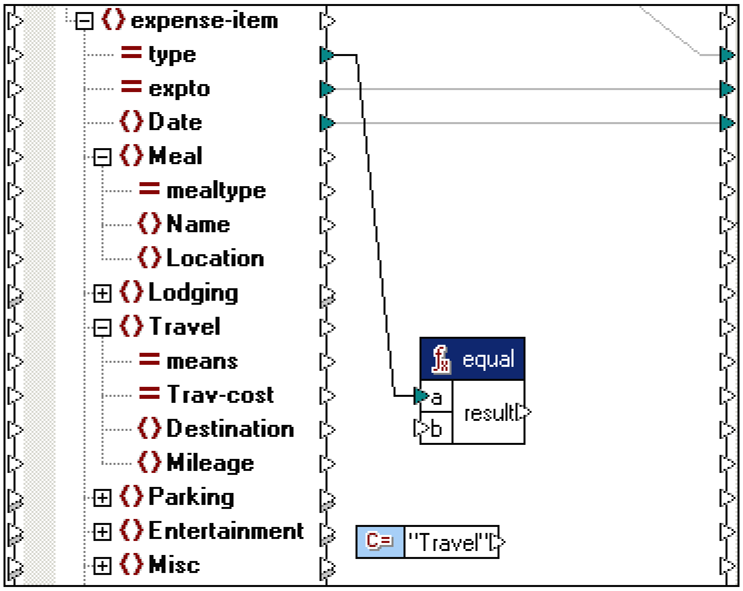
- Using the **Equal** function to test the value of a source item
- Using a **Constant** function to supply the comparison string that is to be tested
- Using the **Filter** function which passes on the Travel data, if the bool input value is true
- Placing the on-true result of this process, into the **expense-item** element of the target schema/document.

Filtering data:

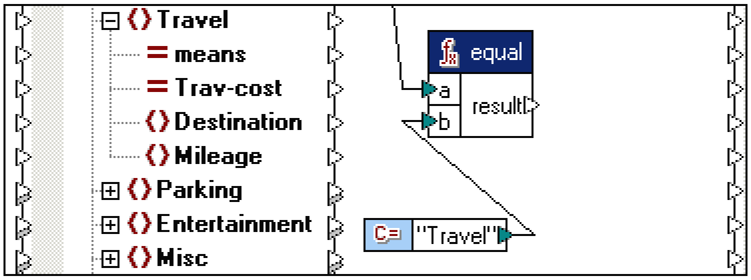
1. Insert a constant component and enter the string **Travel** in the input field.



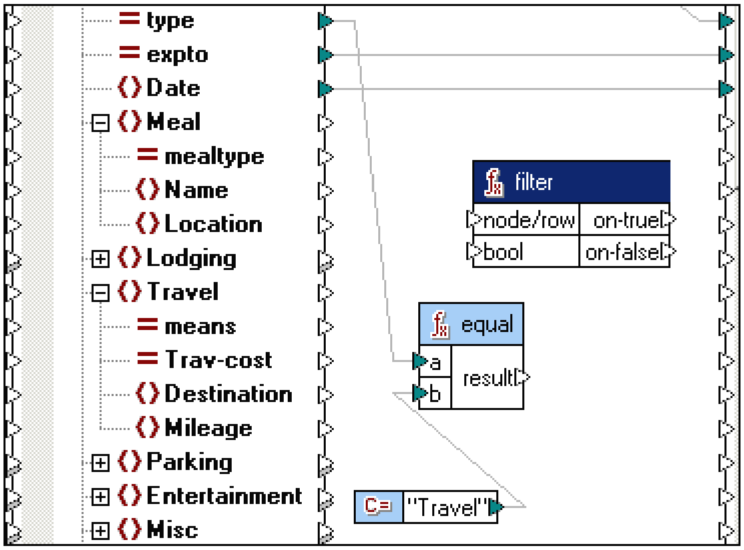
2. Insert the logical function **equal** from the core library (logical functions group).
3. Connect the (expense-item) **type** item in the source schema, to the **a** parameter of the equal function.



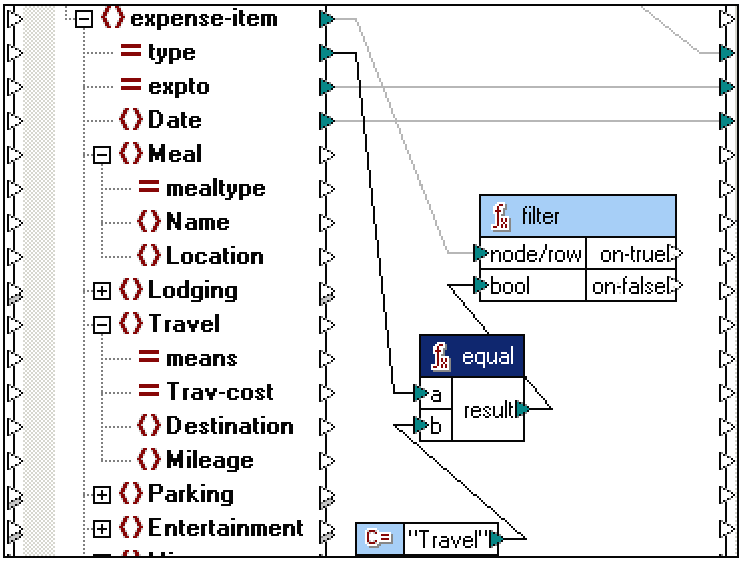
4. Connect the **result** icon of the constant component, to the **b** parameter of the equal function.



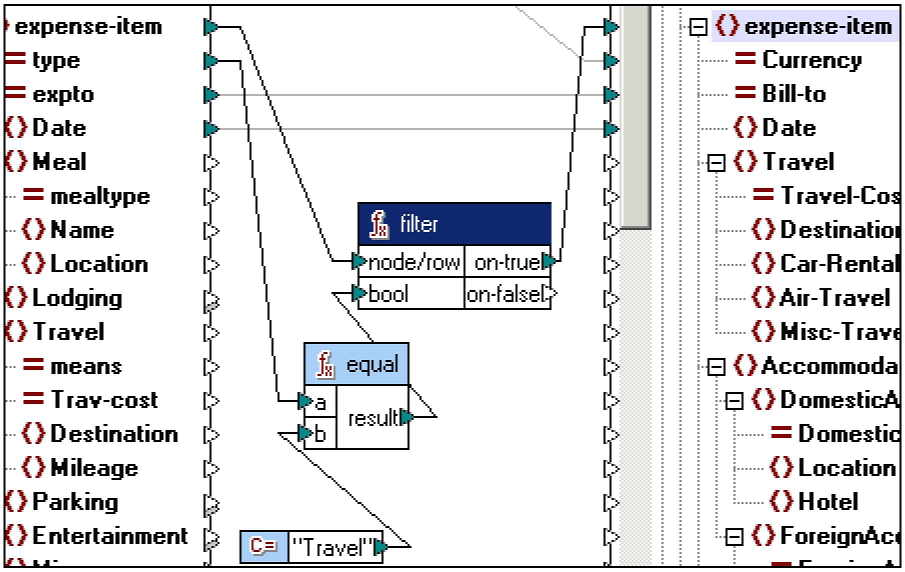
5. Select the menu option **Insert | Filter for Nodes/Rows**.



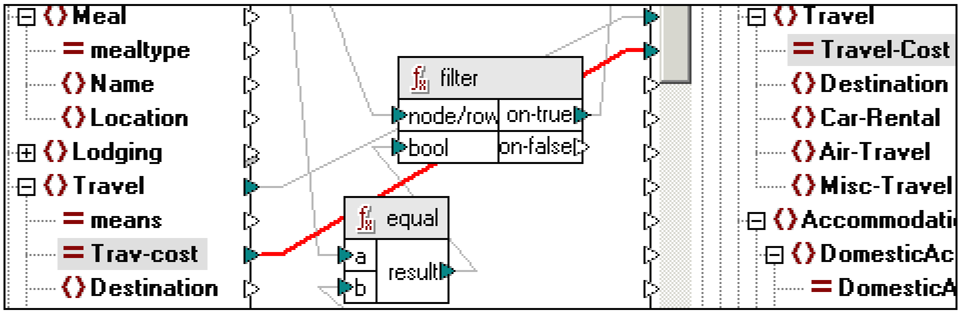
6. Connect the **result** icon of the **equal** component, to the **bool** parameter of the **filter** component.
7. Connect the **expense-item** icon of the source schema with the **node/row** parameter of the filter component.



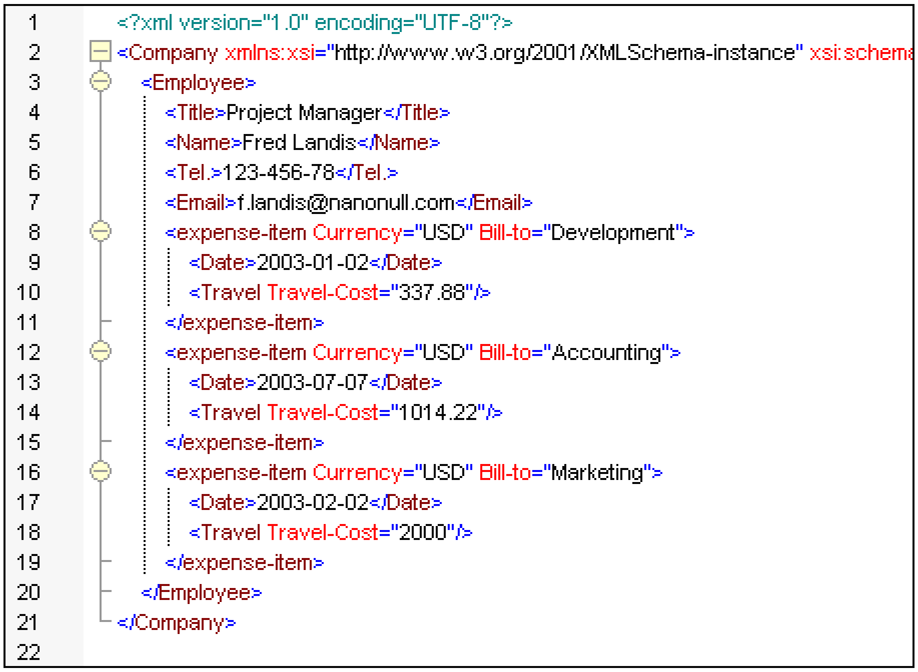
8. Connect the **on-true** icon of the **filter** component with the **expense-item** element of the target document.



9. Connect the **Travel** item in the source schema, with the **Travel** item in the target schema/document.
10. Connect the **Trav-cost** item with the **Travel-Cost** item in the target schema/document.



11. Click the Output tab to see the result.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schema
  <Employee>
    <Title>Project Manager</Title>
    <Name>Fred Landis</Name>
    <Tel.>123-456-78</Tel.>
    <Email>f.landis@nanonull.com</Email>
    <expense-item Currency="USD" Bill-to="Development">
      <Date>2003-01-02</Date>
      <Travel Travel-Cost="337.88"/>
    </expense-item>
    <expense-item Currency="USD" Bill-to="Accounting">
      <Date>2003-07-07</Date>
      <Travel Travel-Cost="1014.22"/>
    </expense-item>
    <expense-item Currency="USD" Bill-to="Marketing">
      <Date>2003-02-02</Date>
      <Travel Travel-Cost="2000"/>
    </expense-item>
  </Employee>
</Company>
```

Please note:

> The **on-false** parameter of the filter component, outputs the **complement** node set that is mapped by the result parameter. In this example it would mean all **non-travel** expense items.

The number of expense-items have been reduced to three. Checking against the supplied **mf-ExpReport.xml** file, reveals that only the Travel records remain, the Lodging and Meal records have been filtered out.

3.5 Generating XSLT 1.0 and 2.0 code

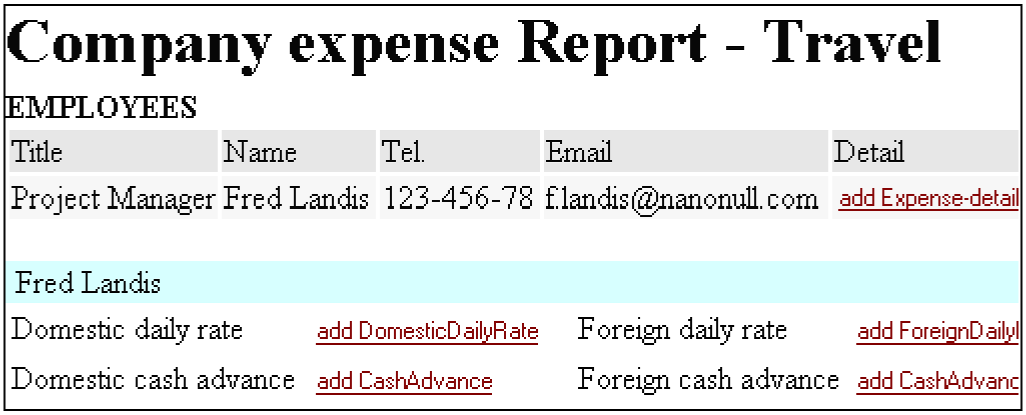
MapForce generates two flavors of XSLT code.

1. Select the menu item **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT file in, and click OK.
   A message appears showing that the generation was successful.
3. Navigate to the designated folder and you will find the XSLT with the file name **MapToExpReport-Target.xslt**

To transform the personal expense report to the company expense report:

Having installed either XMLSpy, or Authentic Desktop you can easily transform the source to the target document.

1. Start XMLSpy, or Authentic Desktop and open the supplied **mf-ExpReport.xml** document.
2. Select the menu option **Tools | Options** and click the **XSL** tab.
3. Enter **.xml** in the *Default file extension of output file field*, and click OK.
4. Select the menu option **XSL/XQuery | XSL Transformation**.
5. Select the previously generated **MapToExpReport-Target.xslt** file, and click OK.
   An XSL Output.xml file is created.
   XMLSpy automatically selects the correct XSLT engine for the transformation.
6. Select the menu option **Authentic | Assign a StyleVision Power Stylesheet**.
7. Select the supplied stylesheet **ExpReport-Target.sps** and click OK.
8. Click the **Authentic** tab to switch to the Authentic view.



Company expense Report - Travel

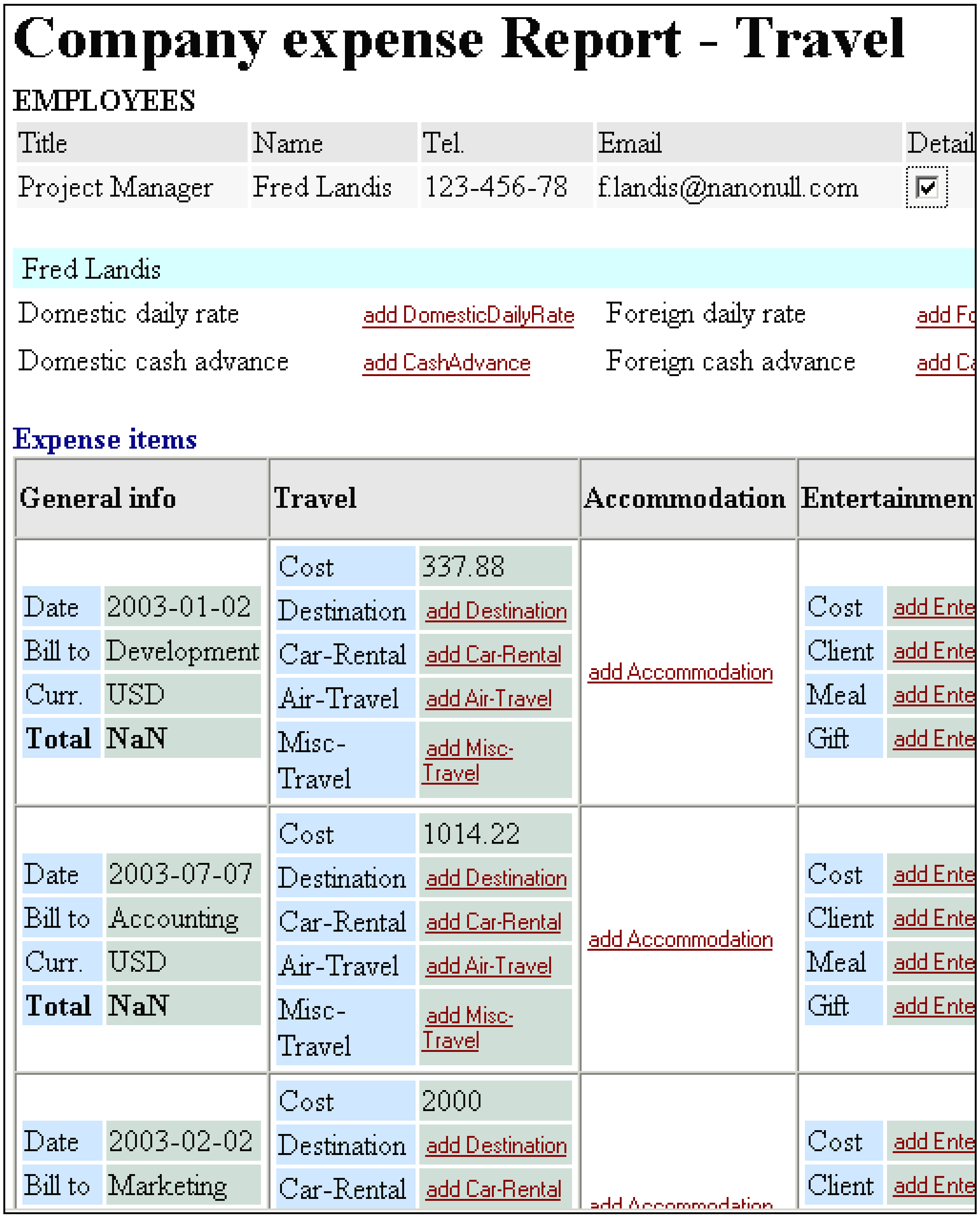
EMPLOYEES

Title	Name	Tel.	Email	Detail
Project Manager	Fred Landis	123-456-78	f.landis@nanonull.com	add Expense-detail

Fred Landis

Domestic daily rate	add DomesticDailyRate	Foreign daily rate	add ForeignDailyI
Domestic cash advance	add CashAdvance	Foreign cash advance	add CashAdvanc

9. Click the **add Expense-detail** text in the Detail column.
   The field changes to a check box.
10. Click the check box to see the detailed expenses.



Company expense Report - Travel

EMPLOYEES

Title	Name	Tel.	Email	Detail
Project Manager	Fred Landis	123-456-78	f.landis@nanonull.com	☑

Fred Landis

Domestic daily rate	add DomesticDailyRate	Foreign daily rate	add Fo
Domestic cash advance	add CashAdvance	Foreign cash advance	add Ca

Expense items

General info	Travel	Accommodation	Entertainmen
Date 2003-01-02 Bill to Development Curr. USD **Total NaN**	Cost 337.88 Destination add Destination Car-Rental add Car-Rental Air-Travel add Air-Travel Misc-Travel add Misc-Travel	add Accommodation	Cost add Ente Client add Ente Meal add Ente Gift add Ente
Date 2003-07-07 Bill to Accounting Curr. USD **Total NaN**	Cost 1014.22 Destination add Destination Car-Rental add Car-Rental Air-Travel add Air-Travel Misc-Travel add Misc-Travel	add Accommodation	Cost add Ente Client add Ente Meal add Ente Gift add Ente
Date 2003-02-02 Bill to Marketing	Cost 2000 Destination add Destination Car-Rental add Car-Rental	add Accommodation	Cost add Ente Client add Ente

The expense report can now be completed with extra information relating to Accommodation, Entertainment and Misc. costs if necessary.

Please note:

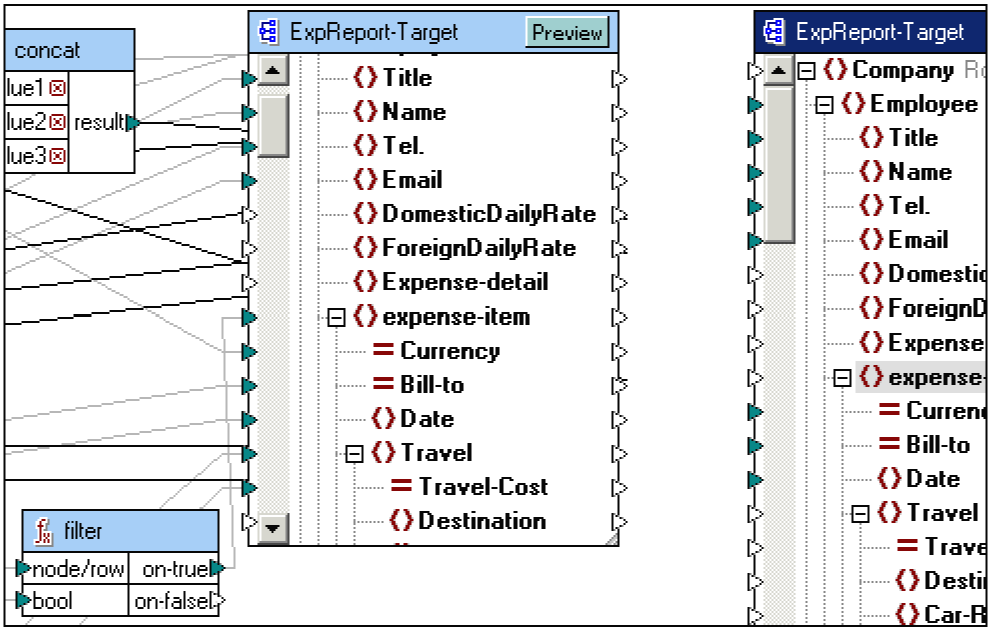
The Total field automatically sums up all Cost fields of each record. Once a number exists in all these fields, the Total field becomes live and the NaN (Not a Number) entry disappears. Subsequent changing of any of the Cost fields, automatically adjusts the Total field.

3.6 Multiple target schemas / documents

This section deals with creating a second target schema / document, into which **non-travel** expense records will be placed, and follows on from the current tutorial example **Tut-ExpReport.mfd**.

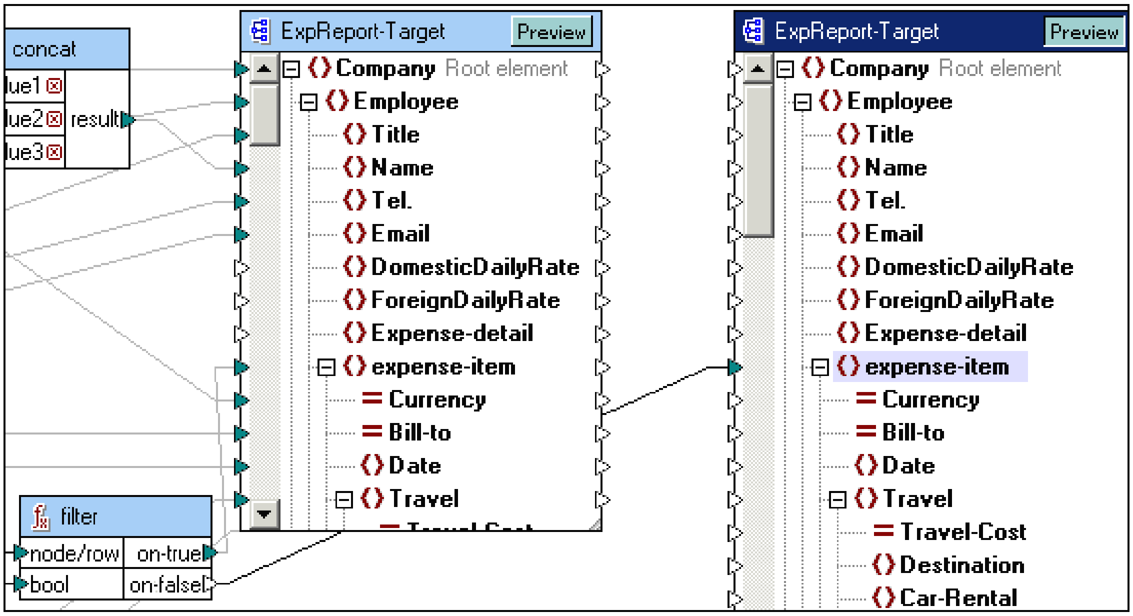
Creating the second target schema component:

1. Click the **Insert XML Schema/File** icon.
2. Select the **ExpReport-Target.xsd** file from the Open dialog box. You are now prompted for a sample XML file for this schema.
3. Click No, and select **Company** as the root element of the target document. The target schema component now appears in the Mapping tab.
4. Click the **Company** entry and hit the * key on the numeric keypad to view all the items.
5. Click the expand window icon and resize the component. Place the schema components so that you can view and work on them easily. There is now one source schema, **mf-expReport**, and two target schemas, both **ExpReport-Target,** visible in the Mapping tab.



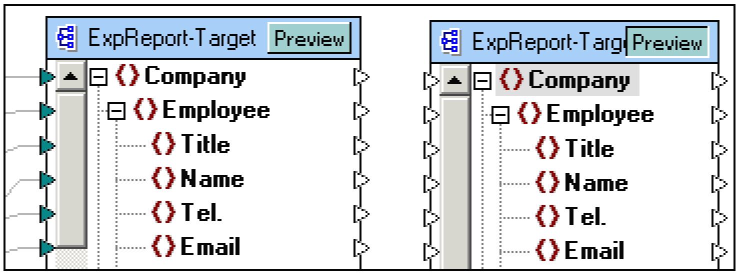
Filtering out the non-travel data:

1. Connect the **on-false** icon of the **filter** component with the **expense-item** element of the **second** target schema / document.



A message appears stating that you are now working with multiple target schemas / documents.

2. Click OK to confirm.



An **Preview icon** is now visible in the title bar of each target schema component.

Clicking the Preview icon defines which of the target schema data is to be displayed, when you subsequently click the XSLT, XSLT2, XQuery, or Output tabs.

Defining multiple target schemas of the same name for code generation:

Both target schemas have the same name in this example, so we have to make sure the code generator can distinguish between them. When generating XSLT there is no need to do this.

1. Right click the **second** target schema/document, and select the Component Settings option.
2. Enter a file name in the **Output XML-instance field**, C:\Progra~1\Altova\MapForce \MapForceExamples\Tutorial\SecondXML.xml for example.

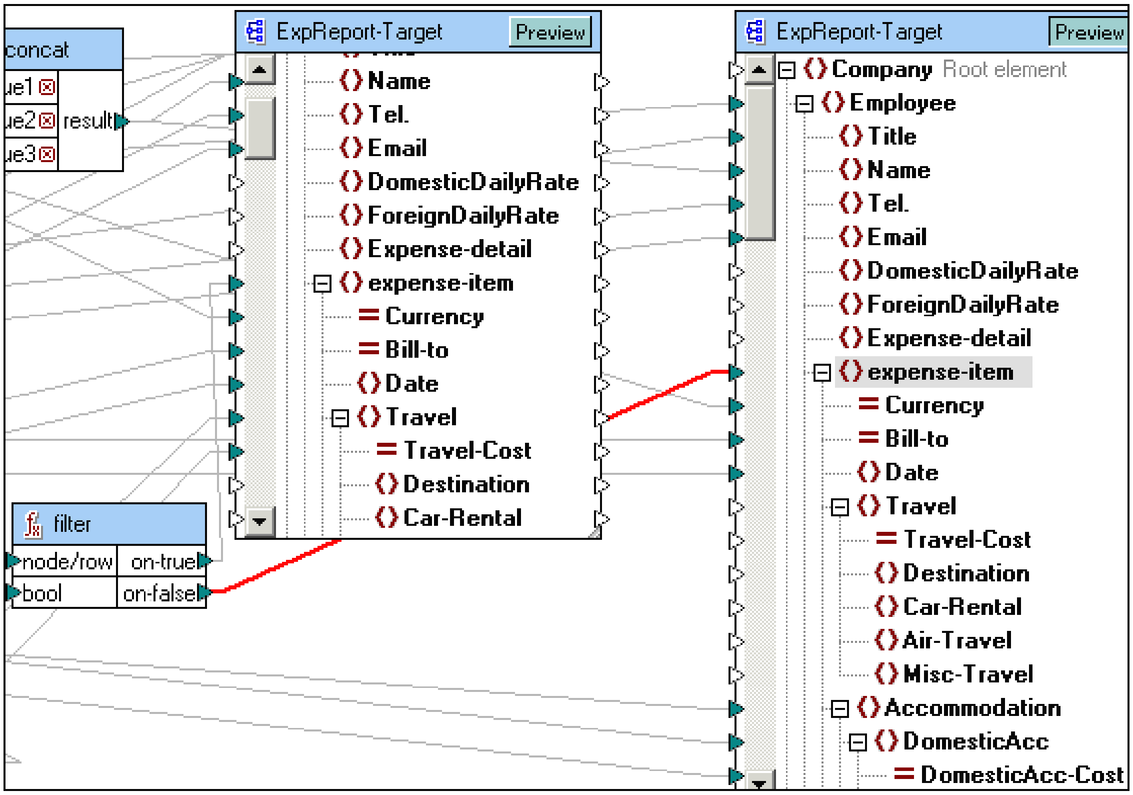
   Note that you have to insert the **absolute path** when generating code. The example above, uses the default installation path of MapForce.

Creating mappings for the rest of the expense report data:

1. Connect the **Lodging** item in the source schema to **Accommodation** in the second target schema.
2. Connect the **Lodging** item to **DomesticAcc**.
3. Connect the **Lodge-Cost** item to **DomesicAcc-Cost**.
4. Create the following mappings between the source schema and second target schema.

You created the same connectors for the first target schema, so there is nothing new here:

Source schema - connect:	**to... second Target schema**
Person	Employee
Result of **existing** First and Last concatenation	Name
Title	Title
Phone	Tel.
Email	Email
currency	Currency
expto	Bill-to
Date	Date

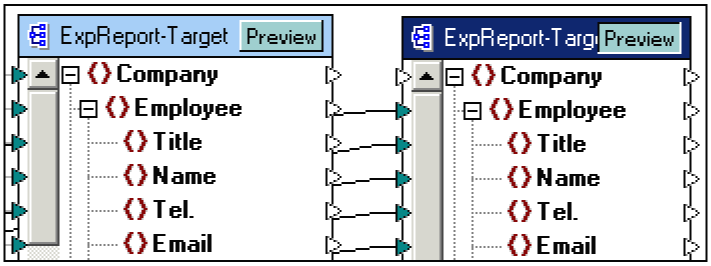


3.6.1 Viewing and generating multiple target schema output

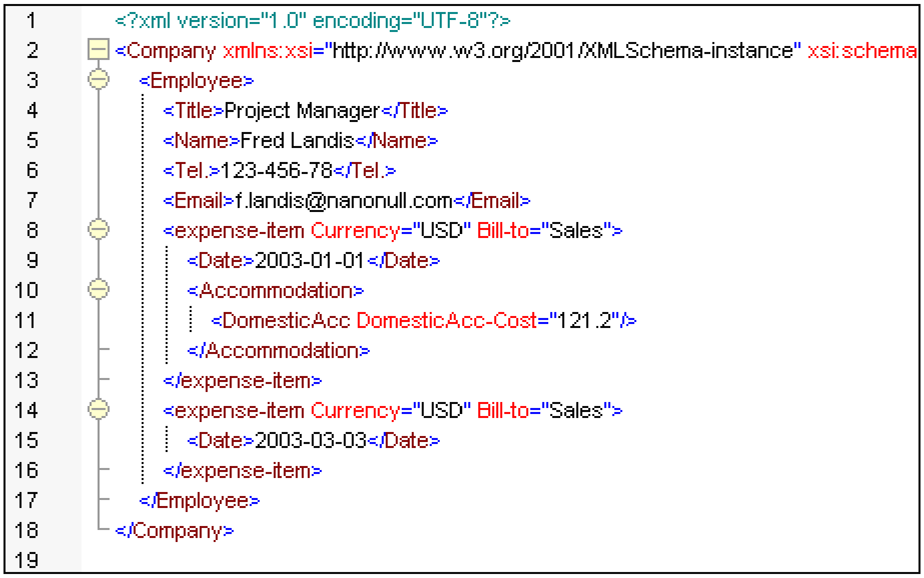
Clicking the Preview icon lets you select which of the schema targets you want to preview.

To view specific XSLT output:

1. Click the **Preview icon** in the title bar of the **second** schema component, to make it active.



2. Click the **Output** tab of the Mapping tab group.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schema
  <Employee>
    <Title>Project Manager</Title>
    <Name>Fred Landis</Name>
    <Tel.>123-456-78</Tel.>
    <Email>f.landis@nanonull.com</Email>
    <expense-item Currency="USD" Bill-to="Sales">
      <Date>2003-01-01</Date>
      <Accommodation>
        <DomesticAcc DomesticAcc-Cost="121.2"/>
      </Accommodation>
    </expense-item>
    <expense-item Currency="USD" Bill-to="Sales">
      <Date>2003-03-03</Date>
    </expense-item>
  </Employee>
</Company>
```

The XML output contains two records both billed to Sales: the Domestic Accommodation cost of $121.2 and an Expense-item record which only contains a date. This record originates from the expense-item Meal. There is currently no mapping between meal costs and domestic accommodation costs, and even if there were, no cost would appear as the XML instance does not supply one.

Please note:

You can save this XML data by clicking the **Save generated output** icon, while viewing the XML output in the preview window .

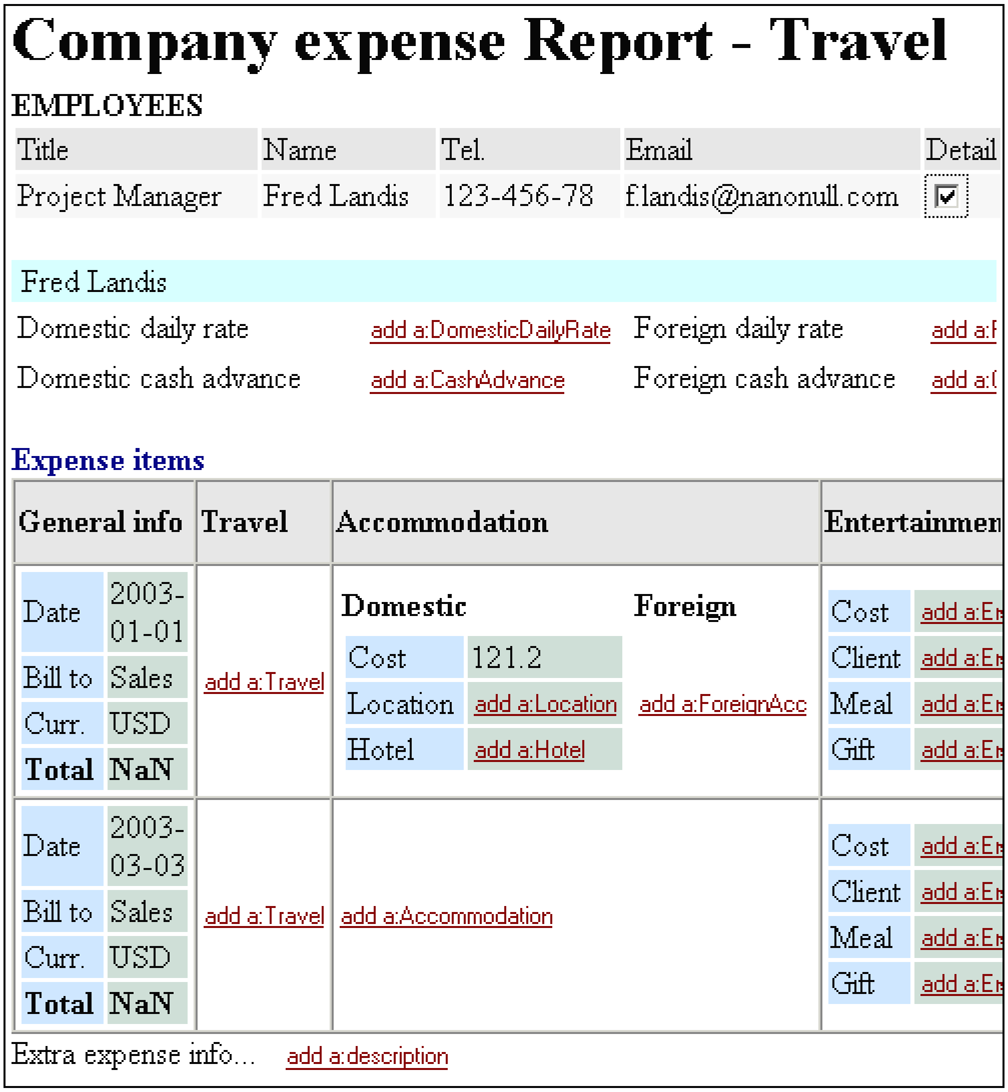
The resulting XML instance file can also be validated against the target schema, by clicking the validate button .

To generate XSLT 1.0 / XSLT 2.0 code for multiple target schemas:

1. Select the menu item **File | Generate code in | XSLT 1.0 (or XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT files, and click OK.
   A message appears showing that the generation was successful.
3. Navigate to the designated folder and you will find two XSLT files with the file names:

MapToExpReport-Target.xslt and **MapToExpReport-Target2.xslt**

4. Having installed either XMLSpy, or Authentic Desktop, assign either of these two XSLT files to the **mf-ExpReport.xml** file, and start the transformation process.
5. Assign the supplied stylesheet, **ExpReport-Target.sps** to the file, and click the Authentic tab.



Company expense Report - Travel

EMPLOYEES

Title	Name	Tel.	Email	Detail
Project Manager	Fred Landis	123-456-78	f.landis@nanonull.com	☑

Fred Landis

Domestic daily rate	add a:DomesticDailyRate	Foreign daily rate	add a:F
Domestic cash advance	add a:CashAdvance	Foreign cash advance	add a:(

Expense items

General info	Travel	Accommodation	Entertainmen
Date 2003-01-01 Bill to Sales Curr. USD **Total NaN**	add a:Travel	**Domestic** Cost 121.2 Location add a:Location Hotel add a:Hotel **Foreign** add a:ForeignAcc	Cost add a:E Client add a:E Meal add a:E Gift add a:E
Date 2003-03-03 Bill to Sales Curr. USD **Total NaN**	add a:Travel	add a:Accommodation	Cost add a:E Client add a:E Meal add a:E Gift add a:E

Extra expense info... add a:description

To generate program code for multiple target schemas:

1. Select the menu item **File | Generate code in | XQuery, Java, C#, or C++**.
2. Select the folder you want to place the generated files in, and click OK. A message appears showing that the generation was successful.
3. Navigate to the designated folder and compile your project.
4. Compile and execute the program code using your specific compiler. Two XML files are generated by the application.

Please note:

A **JBuilder** project file and **Ant** build scripts are generated by MapForce to aid in compiling the Java code, see the section on JDBC driver setup for more information.

3.7 Mapping multiple source items, to single target items

In this section two simple employee travel expense reports will be mapped to a single company report. This example is a simplified version of the mapping you have already worked through in the Multiple target schemas / documents section of this tutorial.

Aim of this section:
To merge two **personal travel expense reports** into a company expense travel report.

Please note that the files used in this example, have been optimized to show how to map data from two input XML files into a single item in the target schema, this is not meant to be a real-life example.

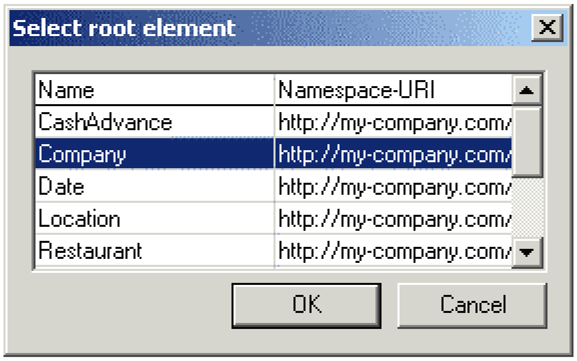
Files used in this section:	
mf-ExpReport.xml	Input XML file used in previous section
mf-ExpReport2.xml	The second input XML file
mf-ExpReport-combined.xml	The resulting file when the mapping has been successful
ExpReport-combined.xsd	The target schema file into which the two XML source data will be merged.
ExpReport-combined.sps	The StyleVision Stylesheet used to view the mapping result in Authentic view.
Tut-ExpReport-msource.mfd	The mapping file for this example

Please note:
The files used in this section are also available in the **...\MapForceExamples\Tutorial** folder.

3.7.1 Creating the mappings

The method described below, is a recapitulation of how to set up the mapping environment.

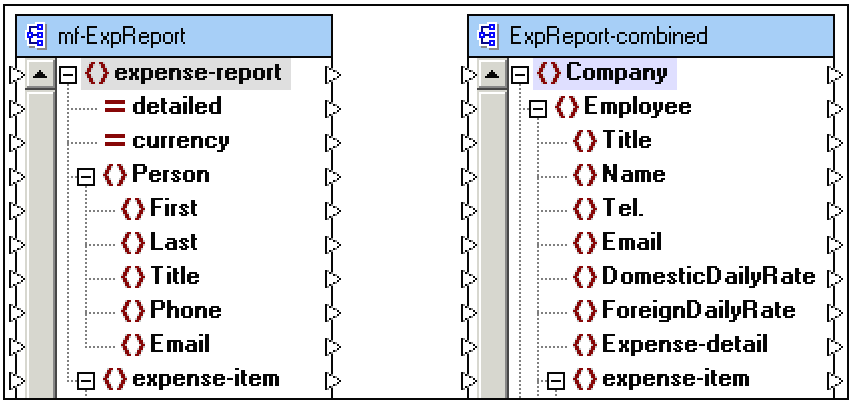
1. Click the **Insert XML Schema/File** icon.
2. Select the **mf-ExpReport.xsd** file from the Open dialog box, and select the **mf-ExpReport.xml** file as the XML instance file.
3. Click the **expense-report** entry, hit the ***** key on the numeric keypad to view all the items; resize the component if necessary.
4. Click the **Insert XML Schema/File** icon.
5. Select the **ExpReport-combined.xsd** file from the Open dialog box. You are now prompted for a sample XML file for this schema.
6. Click No, and select **Company** as the root element of the target document.



Name	Namespace-URI
CashAdvance	http://my-company.com/
Company	http://my-company.com/
Date	http://my-company.com/
Location	http://my-company.com/
Restaurant	http://my-company.com/

The target schema component now appears in the mapping tab.

7. Click the **Company** entry, hit the ***** key on the numeric keypad to view all the items, and resize the window if necessary.



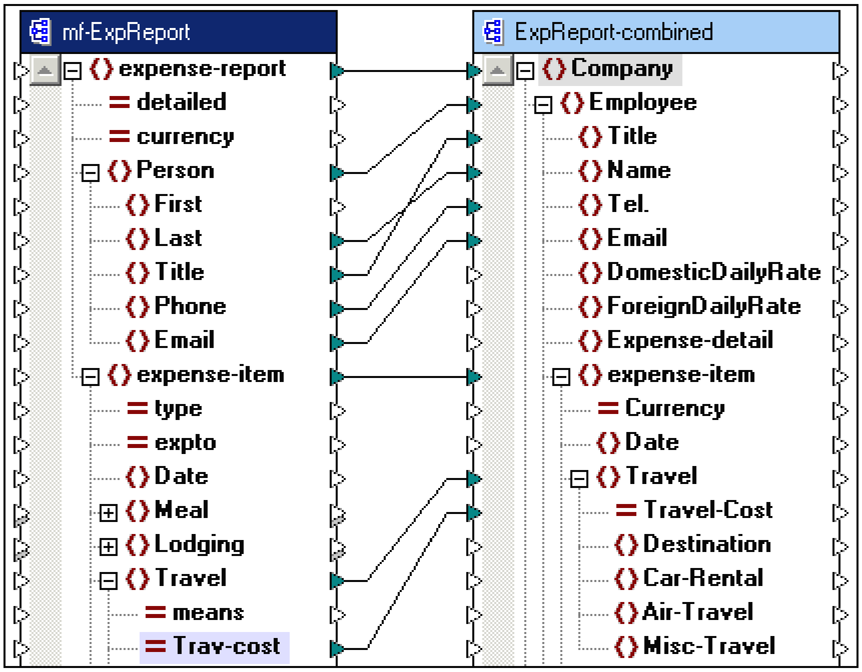
Make sure that the "Auto connect child items" icon is deactivated, before you create the following mappings.

Create the following mappings between the two components:

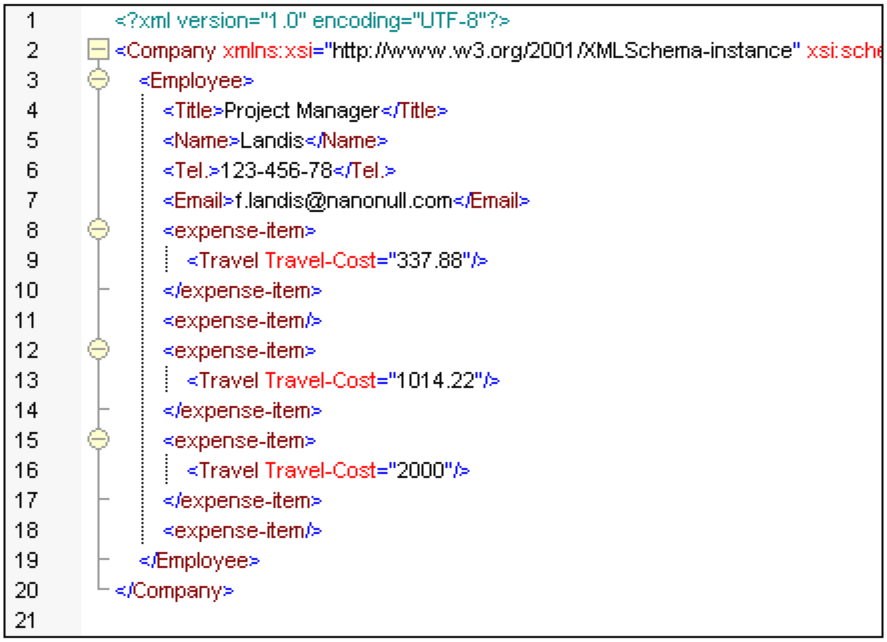
- Expense-report to Company
- Person to Employee
- Last to Name
- Title to Title
- Phone to Tel.
- Email to Email
- expense-item to expense-item
- Travel to Travel and

- Trav-cost to Travel-Cost.

  The mapping is shown below.



8. Click the Output tab to see the result of the current mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sche
  <Employee>
    <Title>Project Manager</Title>
    <Name>Landis</Name>
    <Tel.>123-456-78</Tel.>
    <Email>f.landis@nanonull.com</Email>
    <expense-item>
      <Travel Travel-Cost="337.88"/>
    </expense-item>
    <expense-item/>
    <expense-item>
      <Travel Travel-Cost="1014.22"/>
    </expense-item>
    <expense-item>
      <Travel Travel-Cost="2000"/>
    </expense-item>
    <expense-item/>
  </Employee>
</Company>
```

Please note:

Empty <expense-item/> tags are generated when child items of a **mapped parent item** , exist in the source file, which have not been mapped to the target schema. In this case, only the travel items of the expense-item parent have been mapped. There are however, two other expense items in the list: one lodging and one meal expense item. Each one of these items generates an empty parent expense-item tag.

To avoid generating empty tags, create a filter such as the one described previously in the tutorial, under Filtering data.

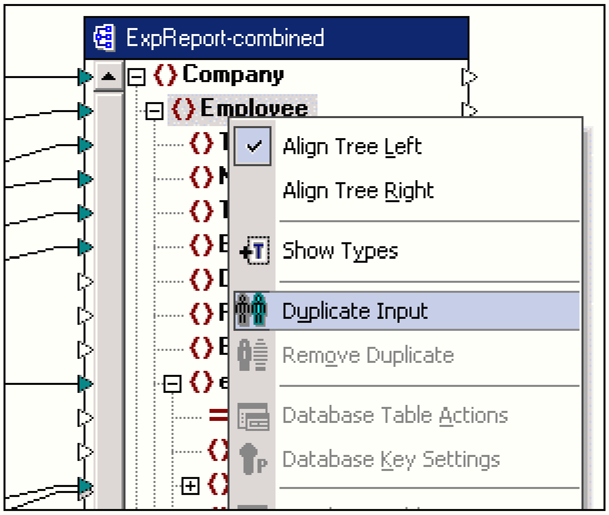
3.7.2 Duplicating input items

We now need to duplicate the **input items** to be able to create mappings from a different source XML file. To achieve this we will:

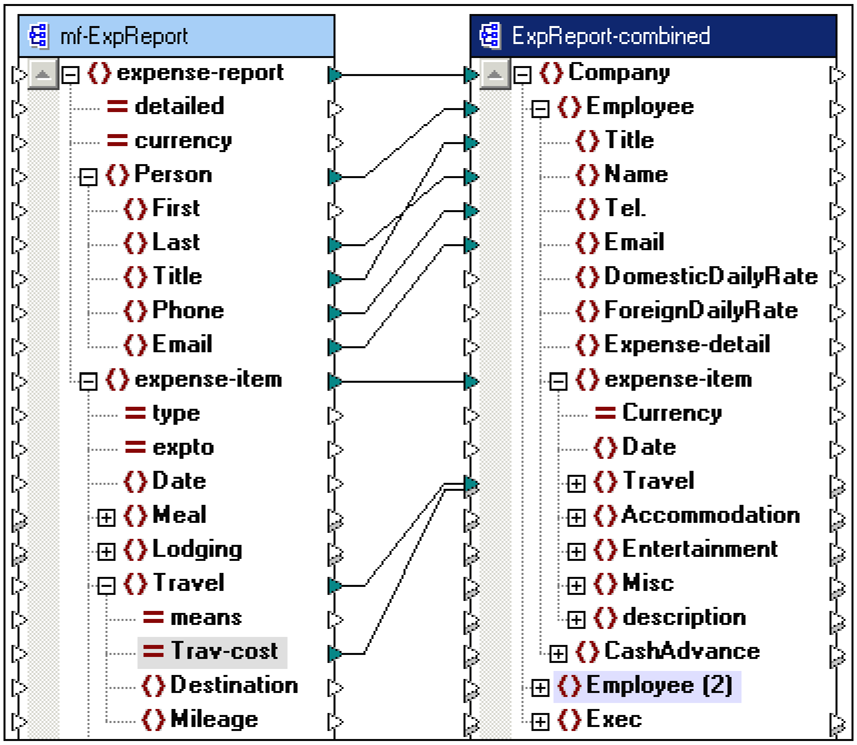
- add the **second** XML source file, and
- create mappings from it, to the "same" inputs in the target XML file.

Duplicating input items:

1. Right click the Employee entry in the target XML file.
2. Select the menu option **Duplicate input.**

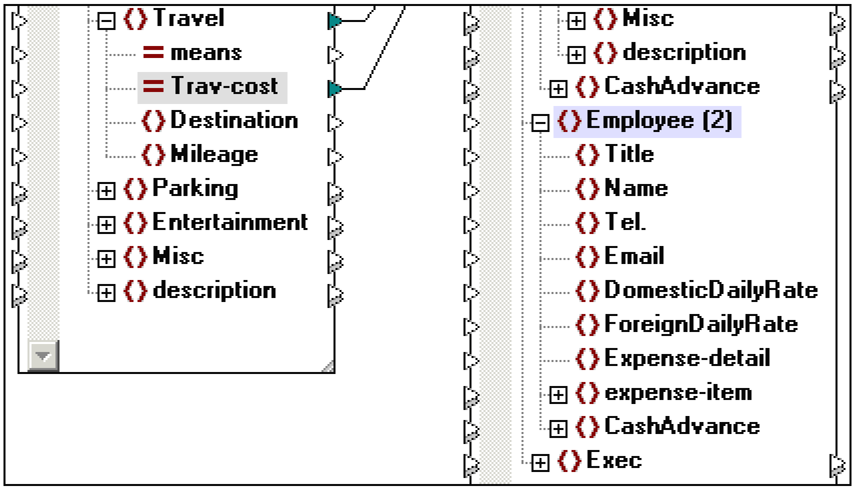


A second Employee item has now been added to the component, as **Employee(2)**.



3. Click the expand icon to see the items below it.

The **structure** of the new Employee item, is an exact copy of the original, except for the fact that there are **no output icons** for the duplicated items.

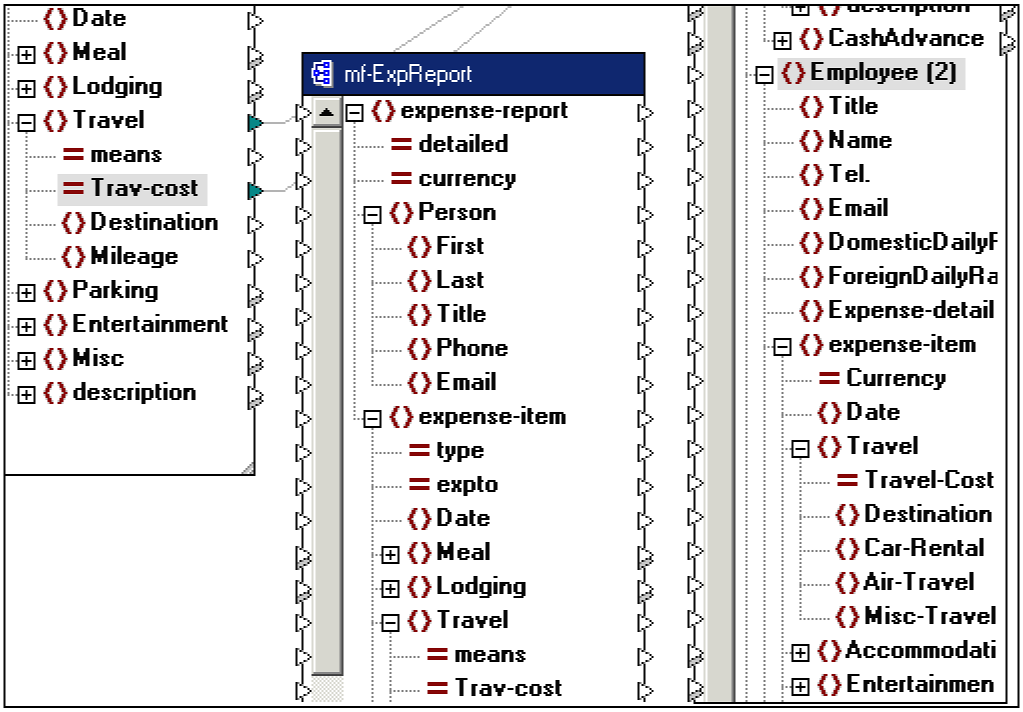


You can now use these new duplicate items as the **target** for the second source XML data file.

Use the same method as before, to insert the second XML instance file:

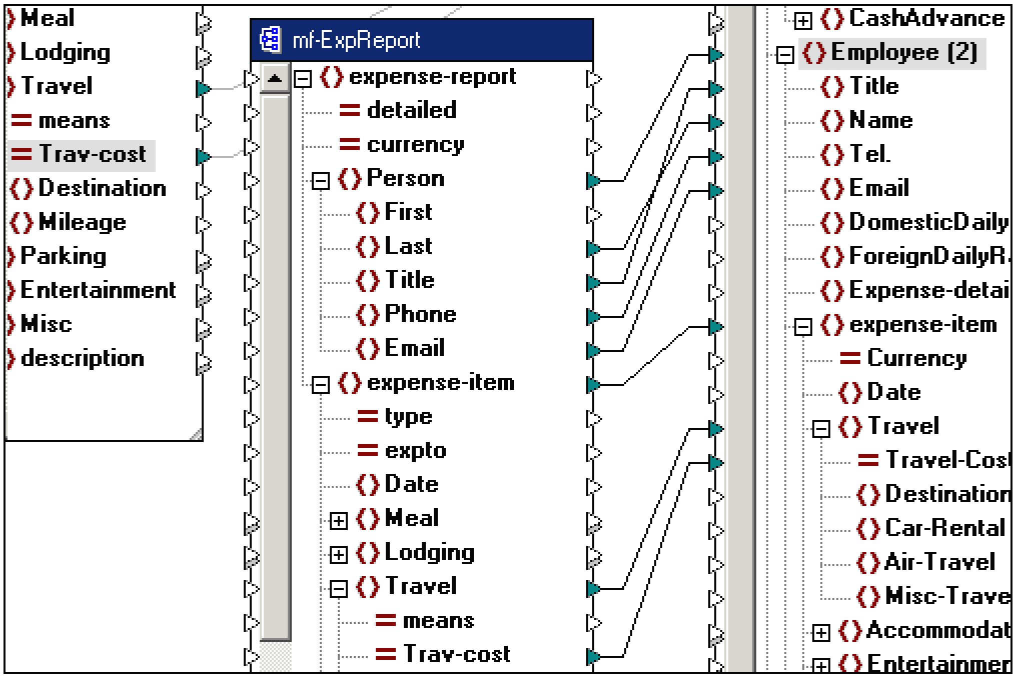
1. Click the **Insert Schema | XML instance** icon.
2. Select the **mf-ExpReport.xsd** file from the Open dialog box, click Yes, and select the **mf-ExpReport2.xml** file as the XML instance file.
3. Click the **expense-report** entry, hit the * key on the numeric keypad to view all items, and resize the component if necessary.

   For the sake of clarity, the new component has been placed between the two existing ones in the following graphics.

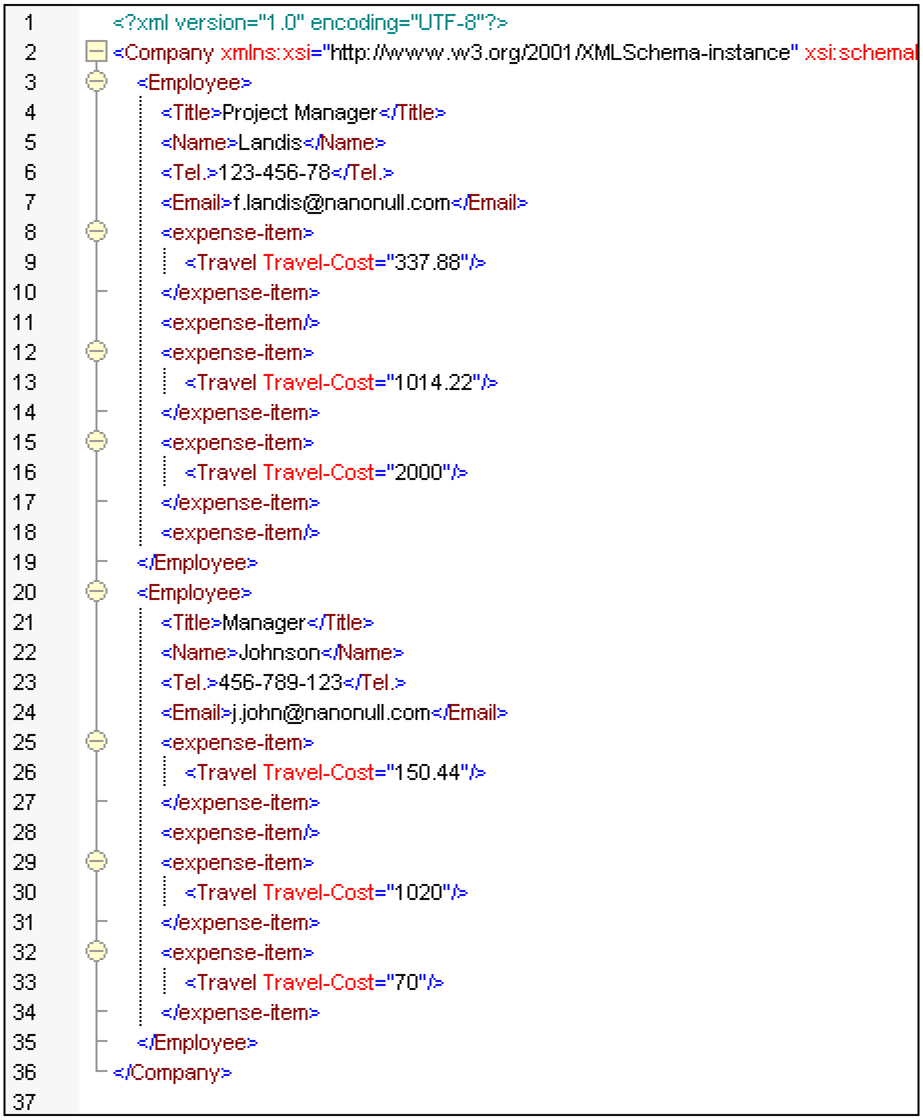


4. Create the same mappings that were defined for the first XML source file:

- Person to Employee(2)
- Last to Name
- Title to Title
- Phone to Tel.
- Email to Email
- expense-item to expense-item
- Scroll down, and map Travel to Travel, and
- Trav-cost to Travel-Cost.



5. Click the Output tab to see the result of the mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemal
  <Employee>
    <Title>Project Manager</Title>
    <Name>Landis</Name>
    <Tel.>123-456-78</Tel.>
    <Email>f.landis@nanonull.com</Email>
    <expense-item>
      <Travel Travel-Cost="337.88"/>
    </expense-item>
    <expense-item/>
    <expense-item>
      <Travel Travel-Cost="1014.22"/>
    </expense-item>
    <expense-item>
      <Travel Travel-Cost="2000"/>
    </expense-item>
    <expense-item/>
  </Employee>
  <Employee>
    <Title>Manager</Title>
    <Name>Johnson</Name>
    <Tel.>456-789-123</Tel.>
    <Email>j.john@nanonull.com</Email>
    <expense-item>
      <Travel Travel-Cost="150.44"/>
    </expense-item>
    <expense-item/>
    <expense-item>
      <Travel Travel-Cost="1020"/>
    </expense-item>
    <expense-item>
      <Travel Travel-Cost="70"/>
    </expense-item>
  </Employee>
</Company>
```

The data of the second expense report has been added to the output file. Johnson and his travel costs have been added to the expense items of Fred Landis in the company expense report.

To save the generated output to a file:

- Click the Save icon which appears in the title bar when the Output tab is active.

The file, mf-ExpReport-combined.xml, is available in the ...\MapforceExamples\Tutorial folder. Please note that it has been assigned an SPS file, which allows you to view the XML file in Authentic View of Authentic Desktop, or XMLSpy.

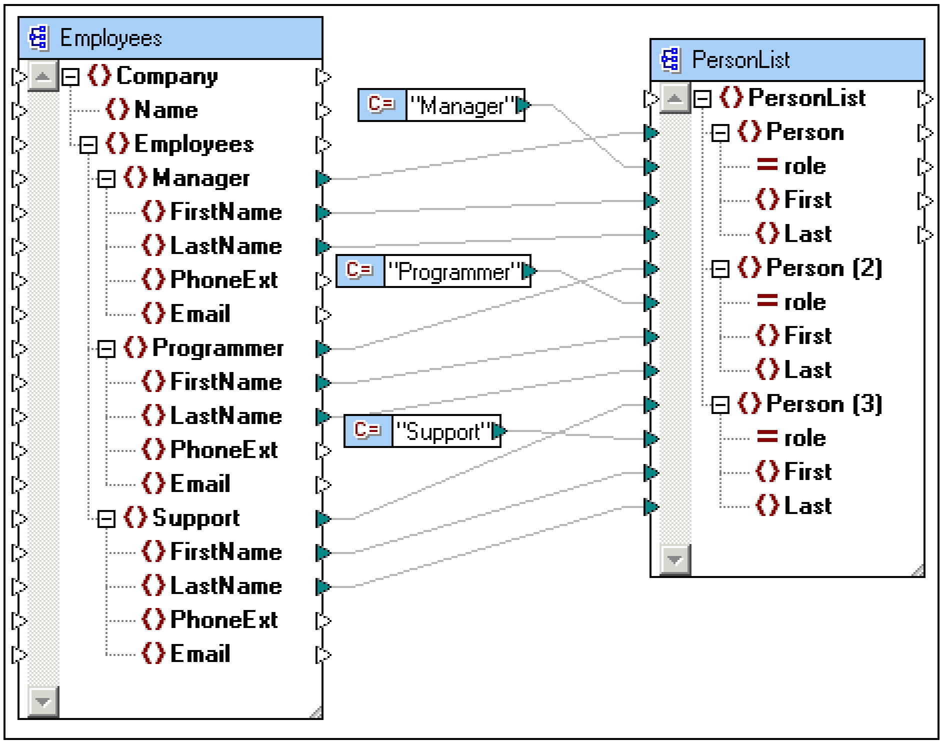
To remove duplicated items:

- Right click the duplicate item and select the **Remove Duplicate** entry from the menu.

To see a further example involving duplicate items, please see the **PersonList.mfd** sample file available in the **...\MapForceExamples** folder.

In the example:

- Different elements of the source document are mapped to the "same" element in the target Schema/XML document.
- Specific elements (Manager etc.) are mapped to a generic one using a "role" attribute.



3.8 Database to schema mapping

This section will show how to use a simple Microsoft Access database as a data source, to map database data to a schema. To use other databases, please see the JDBC driver setup section.

In the current MapForce release, the following databases are supported:

- Microsoft Access 2000 and 2003
- Microsoft SQL Server
- Oracle
- MySQL
- Sybase
- IBM DB2

- ADO compatible databases
- ODBC databases

The table below shows the type of database created, the restrictions, and the connecting methods, when inserting databases.

	Insert Database connection methods (Create "Schema" from Database)		
Supported database	**ODBC restrictions** (unique keys are not supported by ODBC)	**ADO restrictions**	**Oracle (OCI)**
Microsoft Access (ADO)	OK (not recommended) Primary and Foreign keys are not supported.	OK *	-
MS SQL Server (ADO)	OK	OK *	-
Oracle (OCI)	OK, restrictions: table containing columns of type CLOB, BLOB, BFILE; XML tables	OK, restrictions: table containing columns of type CLOB, BLOB, BFILE; XML tables; owner information, Identity constraints are not read from the database	OK *
MySQL (ODBC)	OK *	OK ⌖	-
Sybase (ODBC)	OK *	OK	-
IBM DB2 (ODBC)	OK *	OK	-

* **Recommended connection method for each database.**

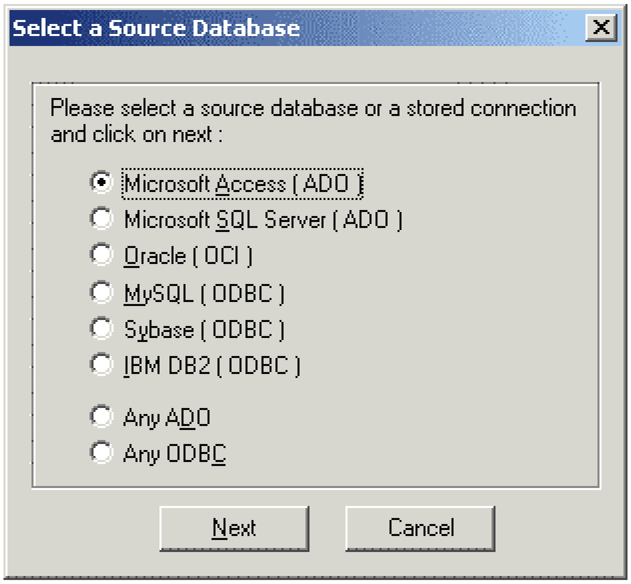
⌖ **MySQL: When creating the ADO connection based on ODBC, it is recommended to use either the User or System DSN**.

- **Not available**

Creating the database component in MapForce:

1. Select **File | New** in MapForce to create a new mapping.

2. Click one of the programming language icons in the title bar: Java, C#, or C++.
3. Click the **Insert Database** icon in the icon bar.



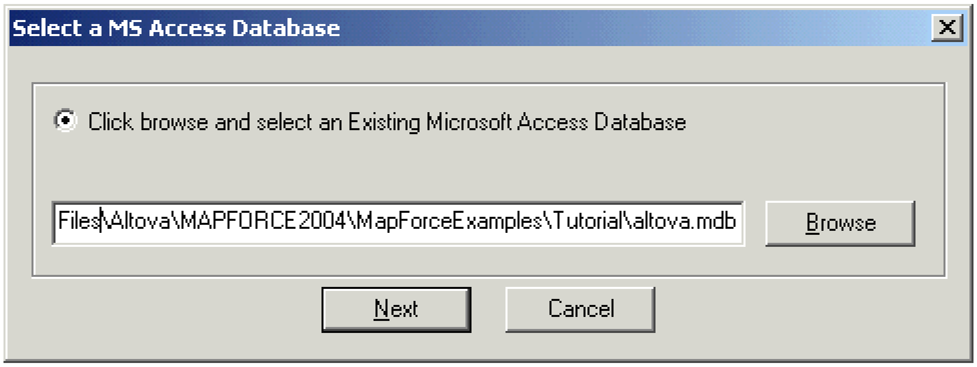
Select a Source Database

Please select a source database or a stored connection and click on next :

● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (OCI)
○ MySQL (ODBC)
○ Sybase (ODBC)
○ IBM DB2 (ODBC)

○ Any ADO
○ Any ODBC

4. Click the **Microsoft Access** radio button.
5. Click the Next button to continue.
6. Click the Browse button to select the database you want as the data source, **altova.mdb** in the ...**\MapForceExamples\Tutorial** folder in this case.
The connection string appears in the text box.



Select a MS Access Database

● Click browse and select an Existing Microsoft Access Database

Files\Altova\MAPFORCE2004\MapForceExamples\Tutorial\altova.mdb

7. Click the **Next** button.
This opens the Create Schema... dialog box.

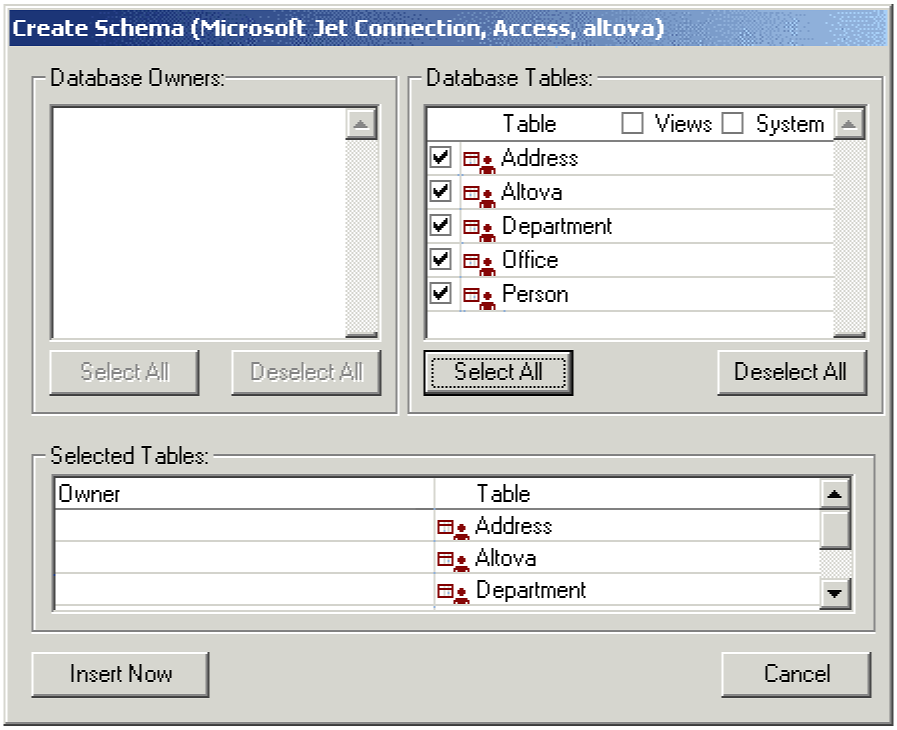
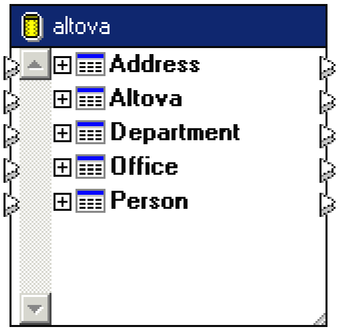


Table	□ Views	□ System
☑ Address		
☑ Altova		
☑ Department		
☑ Office		
☑ Person		

Owner	Table
	Address
	Altova
	Department

8. Click **Select All**, then click the **Insert Now** button to insert the database (schema) component.

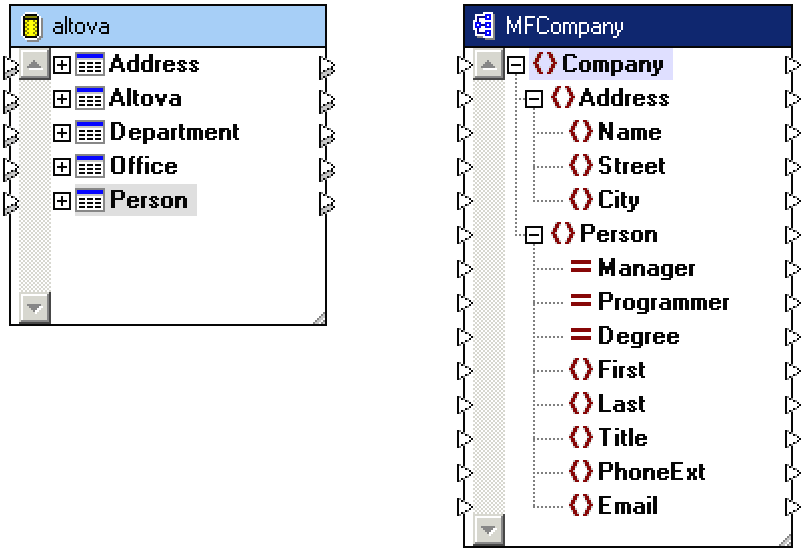


The database component appears in the mapping window. You can now create mappings to a target schema / XML document.

3.8.1 Mapping database data

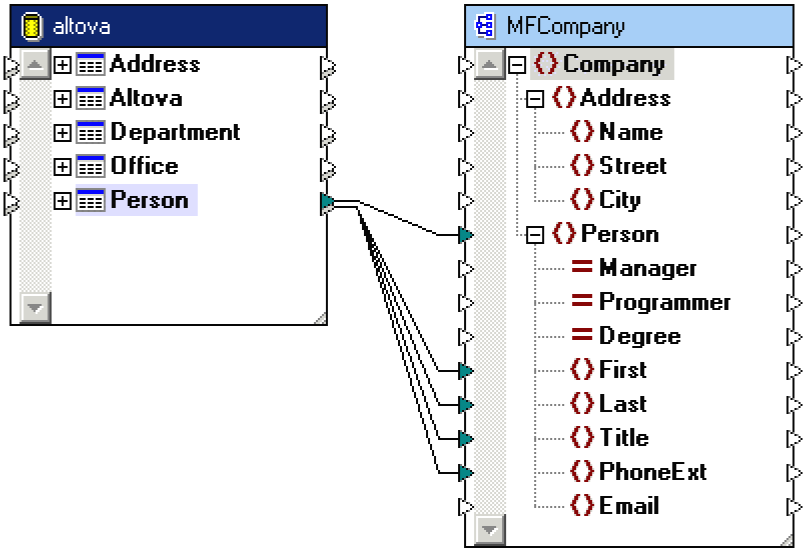
Inserting the target schema /document:

1. Click the **Insert XML Schema/File** icon, and select the **MFCompany.xsd** schema.
2. Click No when the prompt for a sample XML file appears.
3. Select **Company** as the root element and expand all items.
   You are now ready to map the database data to a schema / XML document.



Mapping database data to a schema/document in MapForce:

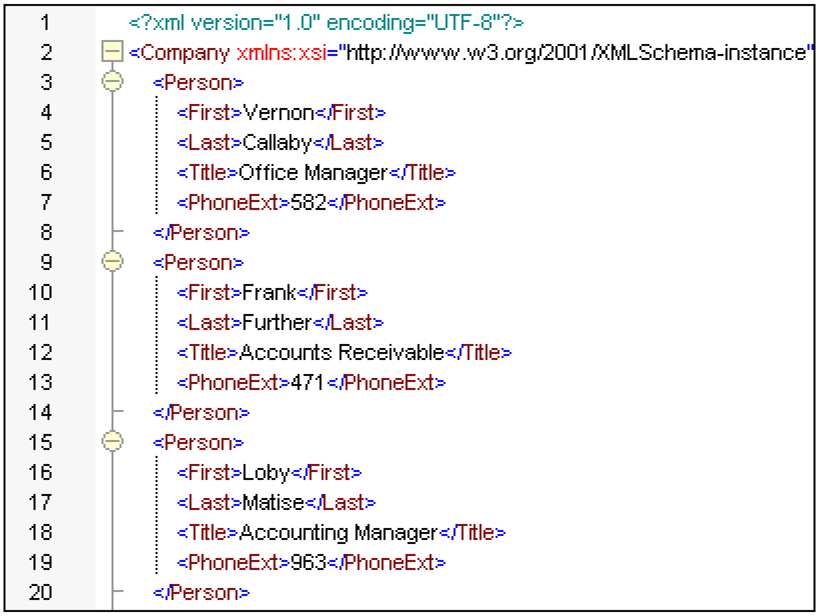
1. Activate the Auto connect child items icon , if not already active.
2. Click the **Person** "table" item in the database component, and connect it to the Person item in MFCompany.
   This creates connectors for all items of the same name in both components.



4. Save the MapForce file, **PersonDB** for example.
5. Click the Output tab to see the result/preview of this mapping. The MapForce engine generates results on-the-fly without you having to generate or compile code.

Generating Java code and the resulting XML file:

1. Select the menu option **File | Generate code in | Java**.
2. Select the directory you want to place the Java files in, and click OK.
   The "Java Code generation completed" message appears when successful.
3. Compile the generated code and execute it.
   The following MFCompany.xml file is created.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Person>
        <First>Vernon</First>
        <Last>Callaby</Last>
        <Title>Office Manager</Title>
        <PhoneExt>582</PhoneExt>
    </Person>
    <Person>
        <First>Frank</First>
        <Last>Further</Last>
        <Title>Accounts Receivable</Title>
        <PhoneExt>471</PhoneExt>
    </Person>
    <Person>
        <First>Loby</First>
        <Last>Matise</Last>
        <Title>Accounting Manager</Title>
        <PhoneExt>963</PhoneExt>
    </Person>
```

For more complex examples of database to schema mapping using:

- multiple source files
- flat and hierarchical databases

Please see the **DB_Altova_SQLXML.mfd** and **DB_Altova_Hierarchical.mfd** files in the **...\MapForceExamples** folder of MapForce.

Chapter 4

Source driven / mixed content mapping

4 Source driven / mixed content mapping

MapForce now supports source driven / mixed content mapping. Source driven / mixed content mapping enables you to automatically map text and child nodes in the same sequence that they appear in the XML source file.

Source-driven mapping can, of course, also be applied to XML schema **complexType** items if you wish. Child nodes will then be mapped according to their sequence in the XML source file.

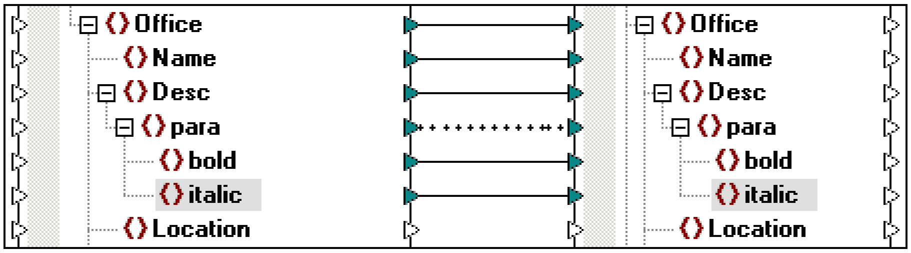
Source driven / mixed content mapping supports:

- XML schema complexTypes as **source** components,
- XML schema complexTypes of type mixed content, i.e. mixed=true, as **source** components,
- XML schema complexTypes (including mixed content), database tables, CSV and fixed-length files, as **target** components

Please note:

Mixed content **text nodes** can only be mapped in their entirety; you cannot limit, or transform the data they contain. Filters, or any other type of function, cannot be used to access text node data.

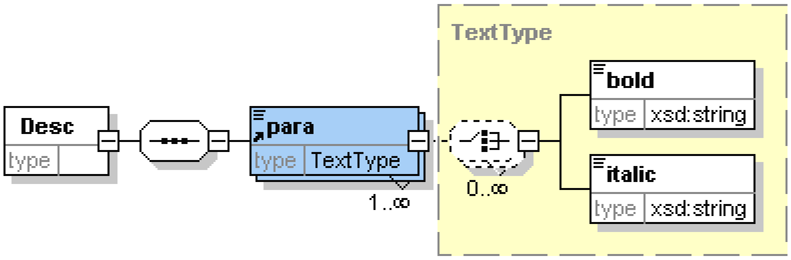
The image below shows an example of mixed content mapping. The para element is of mixed content, and the connector is shown as a dotted line to highlight this.



Right clicking a connector and selecting Connection settings, allows you to annotate, or label the connector. Please see section "Connection" in the Reference section for more information.

The files used in the following example (**Tut-Orgchart.mfd**) are available in the ...\MapForceExamples\Tutorial folder.

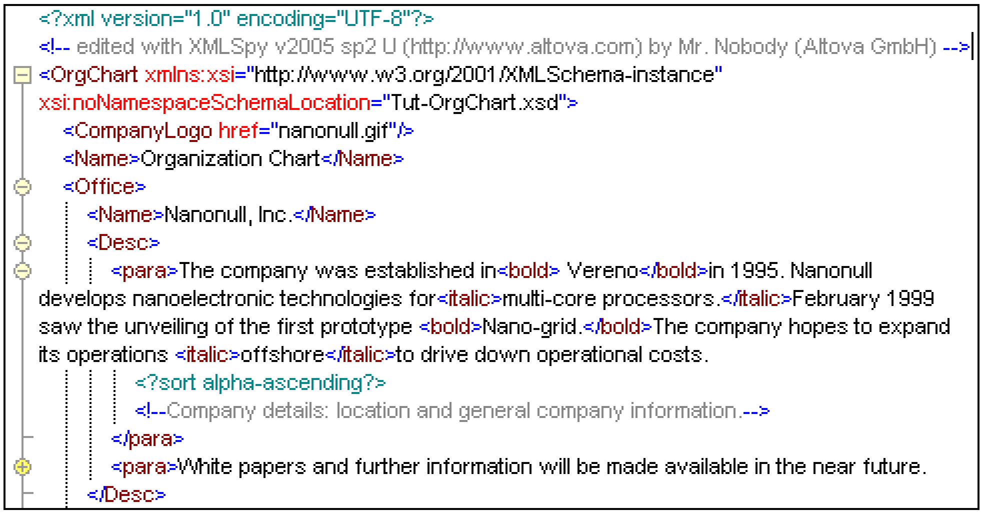
The image below shows the content model of the Description element (Desc) of the **Tut-OrgChart.xsd** schema file. This definition is identical in both the source and target schemas used in this example.



Content model of **para** element:

- para is a complexType with **mixed** = true, of type TextType.
- bold and italic elements are both of type **xsd:string**, they have not been defined as recursive in this example. i.e. neither bold, nor italic are of type "TextType".
- bold and italic elements can appear any number of times in any sequence within para.
- any number of text nodes can appear within the para element, interspersed by any number of bold and italic elements.

Source XML instance:
A portion of the XML file used in this section is shown below. Our area of concern is the mixed content element "para", along with it's child nodes "bold" and "italic". Please note that the para element also contains a Processing Instruction (sort alpha-ascending) as well as Comment text (Company details...) which can also be mapped, see "mixed content settings".



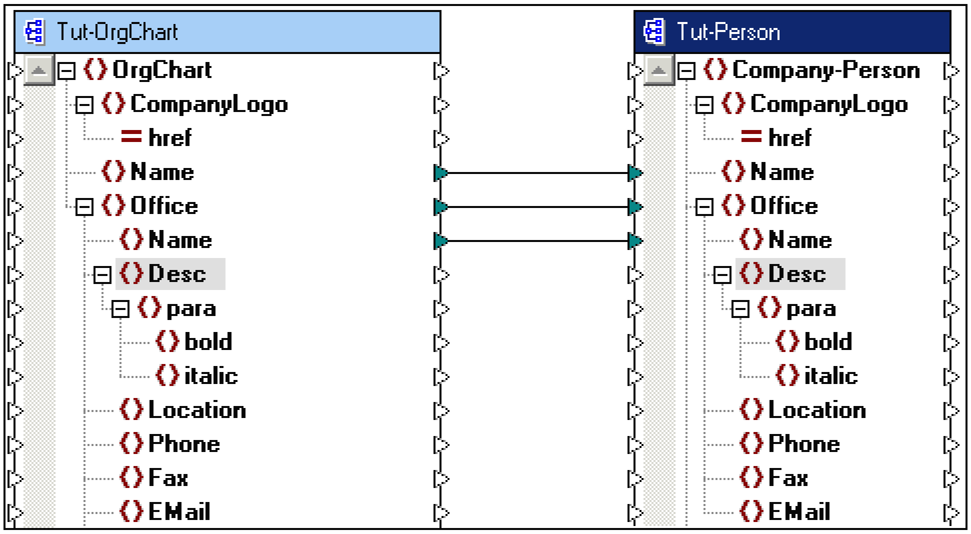
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Mr. Nobody (Altova GmbH) -->
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Tut-OrgChart.xsd">
  <CompanyLogo href="nanonull.gif"/>
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull
develops nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999
saw the unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand
its operations <italic>offshore</italic>to drive down operational costs.
        <?sort alpha-ascending?>
        <!--Company details: location and general company information.-->
      </para>
      <para>White papers and further information will be made available in the near future.
    </Desc>
```

Please note the **sequence** of the text and bold/italic nodes of Nanonull., Inc in the XML instance file, they are:

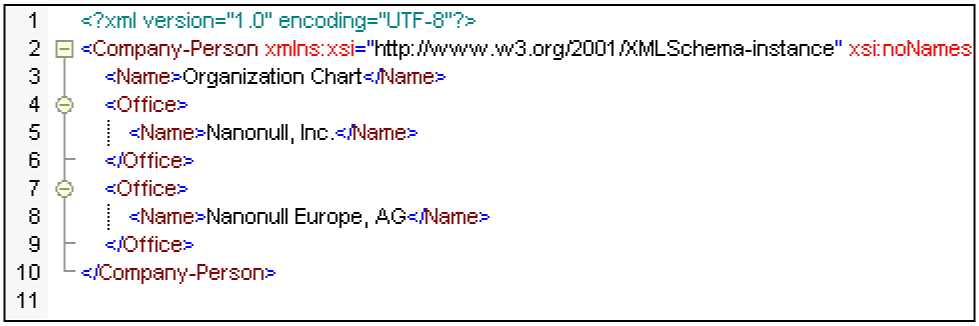
<para> The company...
    <**bold**>Vereno</**bold**>in 1995 ...
    <**italic**>multi-core...</**italic**>February 1999

<**bold**>Nano-grid.<**/bold**>The company ...
<**italic**>offshore...<**/italic**>to drive...
</para>

Mapping
The initial state of the mapping is shown below.



Output of above mapping:
The result of the initial mapping is shown below: Organization Chart as well as the individual office names have been output.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
    </Office>
    <Office>
        <Name>Nanonull Europe, AG</Name>
    </Office>
</Company-Person>
```

4.1 Default settings: mapping mixed content

Creating mixed content connections between items:

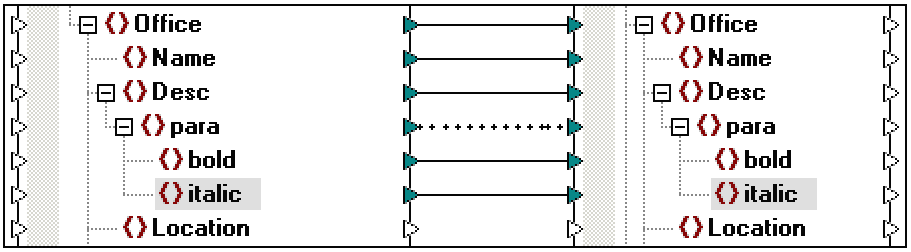
1. Select the menu option **Connection | Auto Connect matching children** to activate this option, if it is not currently activated.
2. Connect the **Desc** item in the source schema, with the **Desc** item in the target schema.

   A message appears, asking if you would like to create a mixed content connection. You are also notified that the text and child items will be transferred in the same order they appear in the XML source file.
3. Click Yes to create a mixed content connector.

Please note:

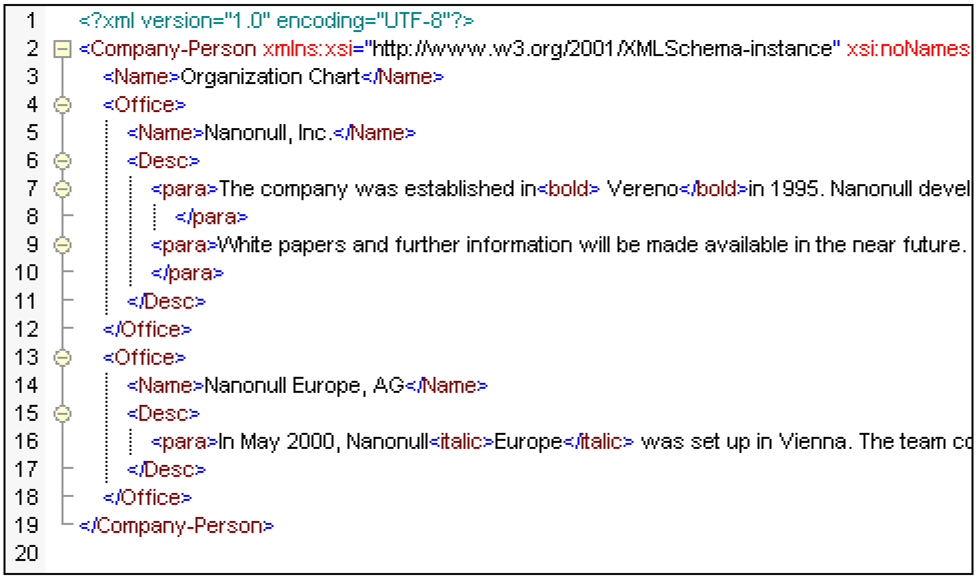
Although the Desc is not of mixed content, a message appears because the auto-connect option has been activated, and para exists in both source and target components. Para is of content, and makes the message appear at this point.

The mixed-content message also appears if you only map the para items directly, without having the autoconnect option activated.



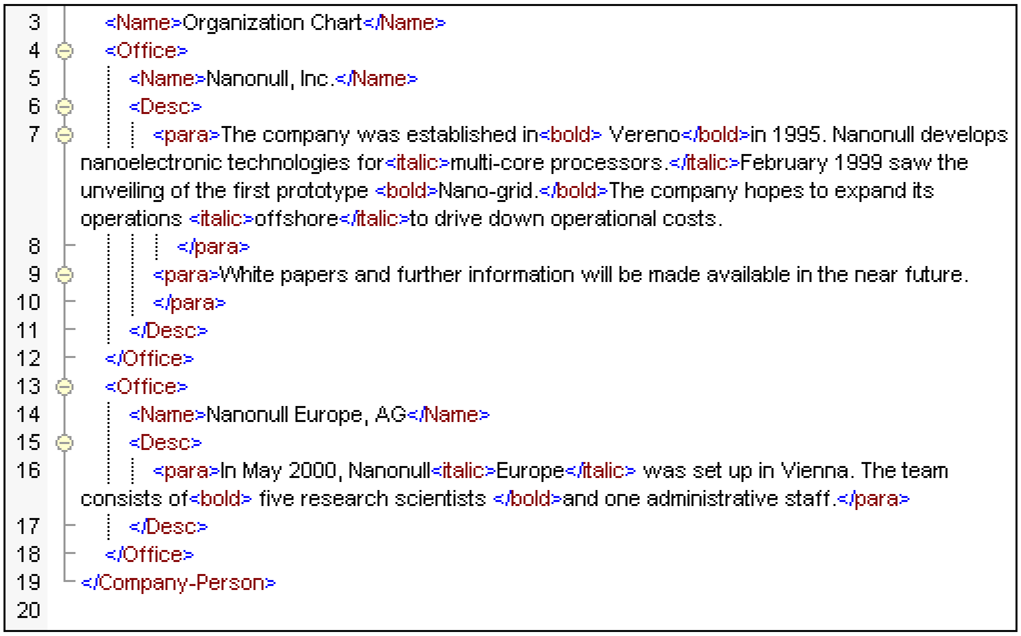
All child items of Desc have been connected. The connector joining the para items is displayed as a dotted line, to show that it is mixed content.

4. Click the Output tab to see the result of the mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
        <Desc>
            <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel
                </para>
            <para>White papers and further information will be made available in the near future.
            </para>
        </Desc>
    </Office>
    <Office>
        <Name>Nanonull Europe, AG</Name>
        <Desc>
            <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team co
        </Desc>
    </Office>
</Company-Person>
```

5. Click the word **Wrap** icon , to view display the complete text in the Output window.



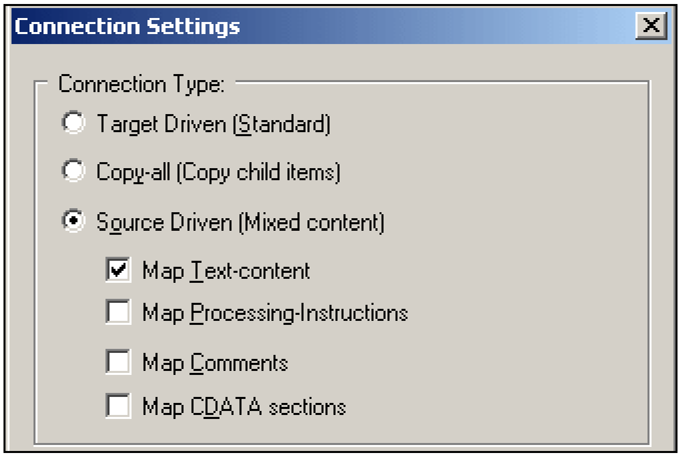
```
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
        <Desc>
            <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand its
operations <italic>offshore</italic>to drive down operational costs.
                </para>
            <para>White papers and further information will be made available in the near future.
            </para>
        </Desc>
    </Office>
    <Office>
        <Name>Nanonull Europe, AG</Name>
        <Desc>
            <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team
consists of<bold> five research scientists </bold>and one administrative staff.</para>
        </Desc>
    </Office>
</Company-Person>
```

The mixed content text of each office description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML source file.

6. Switch back to the Mapping view.

Removing text nodes from mixed content items:

1. Right click the para connector and select **Connection Settings**.



Connection Settings

Connection Type:

○ Target Driven (Standard)

○ Copy-all (Copy child items)

● Source Driven (Mixed content)

☑ Map Text-content

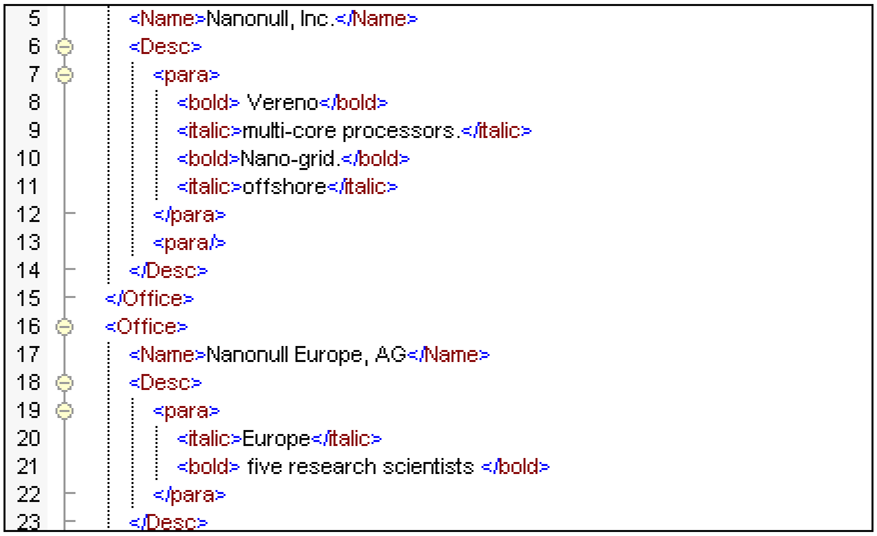
☐ Map Processing-Instructions

☐ Map Comments

☐ Map CDATA sections

The image shows the default settings when you first create mixed content mapping. The "Map Text content" check box is active per default.

2. **Deactivate** the Map Text content check box and click OK to confirm.
3. Click the Output tab to see the result of the mapping.



```
    <Name>Nanonull, Inc.</Name>
    <Desc>
        <para>
            <bold> Vereno</bold>
            <italic>multi-core processors.</italic>
            <bold>Nano-grid.</bold>
            <italic>offshore</italic>
        </para>
        <para/>
    </Desc>
</Office>
<Office>
    <Name>Nanonull Europe, AG</Name>
    <Desc>
        <para>
            <italic>Europe</italic>
            <bold> five research scientists </bold>
        </para>
    </Desc>
```

Result:

- all **text** nodes of the para element have been removed.
- mapped bold and italic text content remain
- bold and italic item **sequence** still follow that of the source XML file!

Text nodes and mixed content mapping:

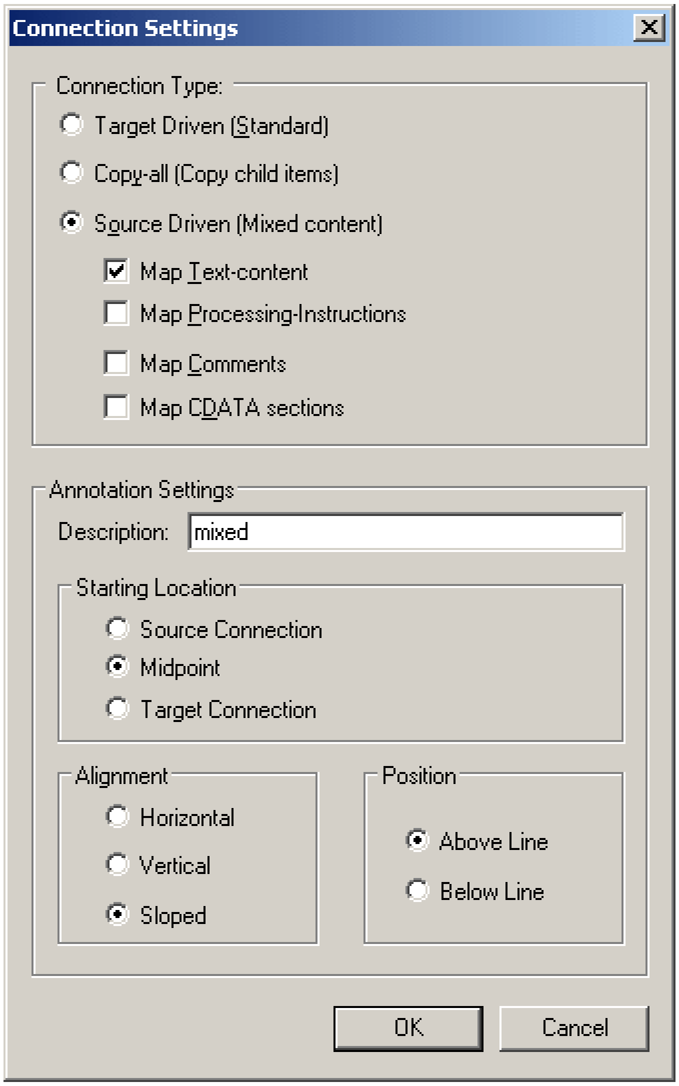
- Text nodes can only be mapped in their entirety; you cannot limit, or transform the data they contain. All text nodes of the para element are either mapped, or excluded, as in the example above.
- Filters, or any other type of function, cannot be used to access text node data.
- Mixed content child node data, i.e. data enclosed in bold/italic tags in this example, can of course be mapped individually. If a connector exists, then the child data will be mapped.
- There is currently no way of accessing the text node(s) of a mixed content element, for further processing, or filtering.

Mixed content settings:

- Right click the para connector and select Connection Settings.

  This opens the Connection Settings dialog box in which you can define the specific (mixed content) settings of the current connector. Note that unavailable options are greyed out.

  Please note that these settings also apply to **complexType** items which do not have any text nodes!



Connection Settings

Connection Type:

○ Target Driven (Standard)

○ Copy-all (Copy child items)

● Source Driven (Mixed content)

☑ Map Text-content

□ Map Processing-Instructions

□ Map Comments

□ Map CDATA sections

Annotation Settings

Description: mixed

Starting Location

○ Source Connection

● Midpoint

○ Target Connection

Alignment

○ Horizontal

○ Vertical

● Sloped

Position

● Above Line

○ Below Line

Target Driven (Standard)
Changes the connector type to Standard mapping, please see: "Source-driven / mixed content vs. standard mapping" for more information.

Source Driven (mixed content)
Changes the connector type to source driven / mixed content, and enables the selection of additional elements to be mapped. The additional elements have to be **child items** of the mapped item in the XML source file, to be able to be mapped.

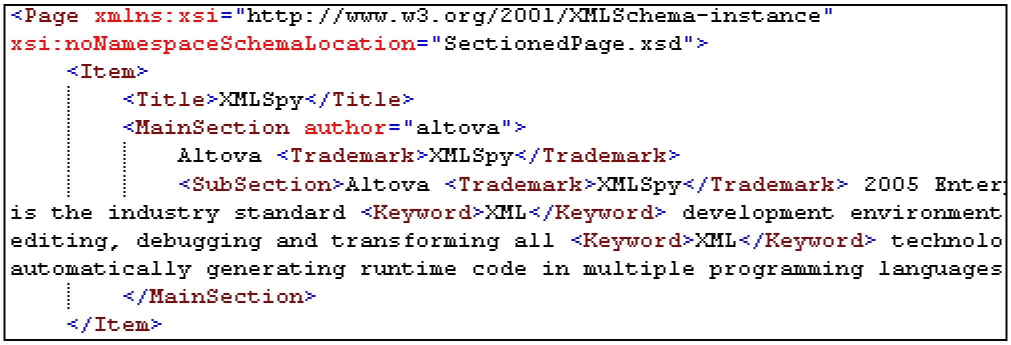
Annotation settings:
Individual connectors can be labeled for clarity.

1. Double click a connector and enter the name of the connector in the Description field. This enables all the options in the Annotation Settings group.
2. Use the remaining groups to define the position and alignment of the label.

4.2 Mixed content example

The following example is available as "**ShortApplicationInfo.mfd**" in the ...\MapForceExamples folder.

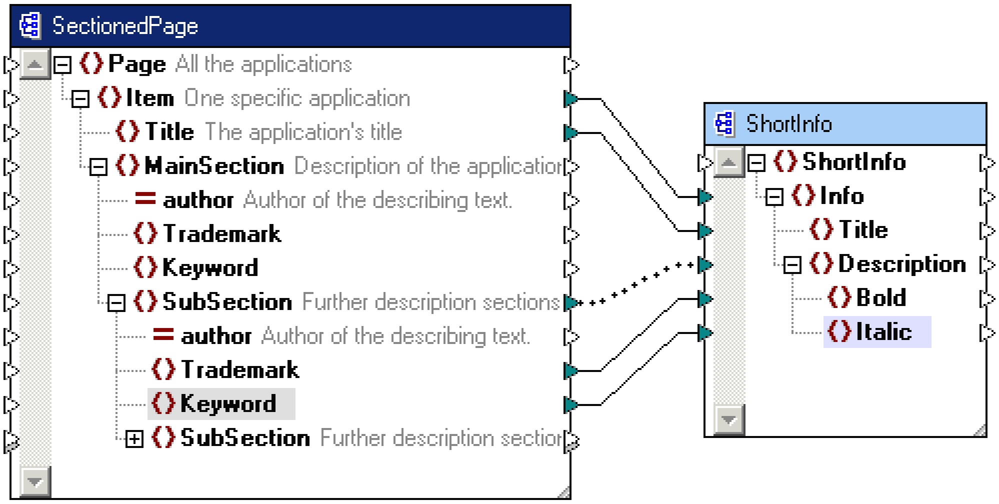
A snippet of the XML source file for this example is shown below.



```
<Page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SectionedPage.xsd">
    <Item>
        <Title>XMLSpy</Title>
        <MainSection author="altova">
            Altova <Trademark>XMLSpy</Trademark>
            <SubSection>Altova <Trademark>XMLSpy</Trademark> 2005 Enter
is the industry standard <Keyword>XML</Keyword> development environment
editing, debugging and transforming all <Keyword>XML</Keyword> technolo
automatically generating runtime code in multiple programming languages
        </MainSection>
    </Item>
```

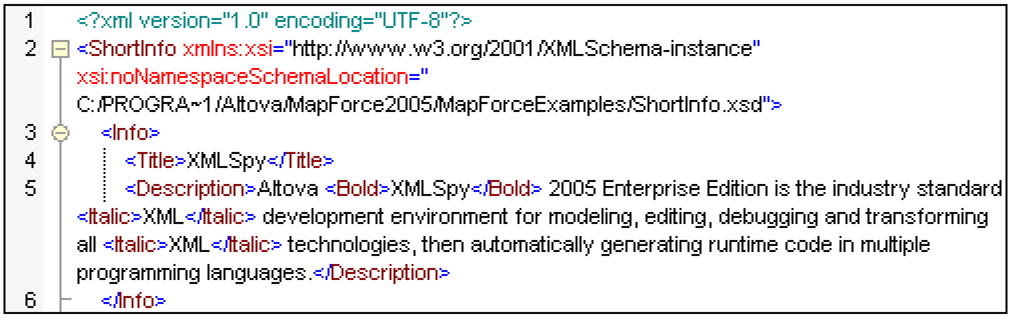
The mapping is shown below. Please note that:

- The Subsection item connector is of mixed content, and is mapped to the Description item in the target XML/schema.
- Trademark text is mapped to the Bold item in the target
- Keyword text is mapped to the Italic item in the target


SectionedPage
Page All the applications
Item One specific application
Title The application's title
MainSection Description of the application
author Author of the describing text.
Trademark
Keyword
SubSection Further description sections
author Author of the describing text.
Trademark
Keyword
SubSection Further description section
ShortInfo
ShortInfo
Info
Title
Description
Bold
Italic

Mapping result:

- The mixed content text of each description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML source file.



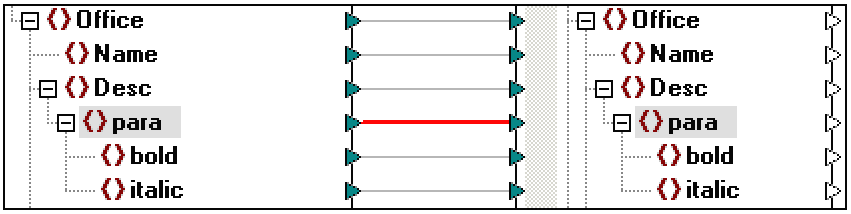
```
<?xml version="1.0" encoding="UTF-8"?>
<ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="
C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
    <Info>
        <Title>XMLSpy</Title>
        <Description>Altova <Bold>XMLSpy</Bold> 2005 Enterprise Edition is the industry standard
<Italic>XML</Italic> development environment for modeling, editing, debugging and transforming
all <Italic>XML</Italic> technologies, then automatically generating runtime code in multiple
programming languages.</Description>
    </Info>
```

4.3 Source-driven / mixed content vs. standard mapping

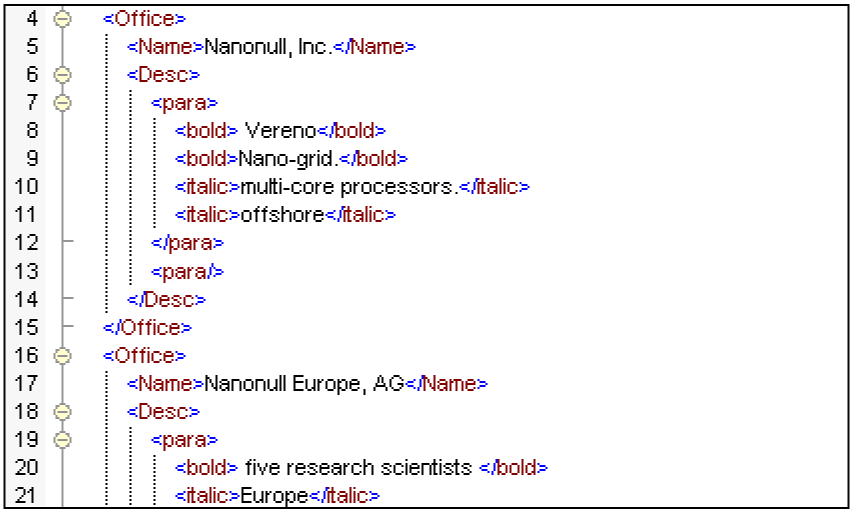
This section describes the results when defining standard mappings (or using standard connectors) on mixed content items. The files used in the following example (**Tut-Orgchart.mfd**) are available in the ...\MapForceExamples\Tutorial folder.

Creating standard connections between mixed content items:

1. Right click the para connector and select **Target Driven (Standard)** from the popup window.
   The connector now appears as a solid line.



2. Click the Output tab to see the result of the mapping.



```
<Office>
  <Name>Nanonull, Inc.</Name>
  <Desc>
    <para>
      <bold> Vereno</bold>
      <bold>Nano-grid.</bold>
      <italic>multi-core processors.</italic>
      <italic>offshore</italic>
    </para>
    <para/>
  </Desc>
</Office>
<Office>
  <Name>Nanonull Europe, AG</Name>
  <Desc>
    <para>
      <bold> five research scientists </bold>
      <italic>Europe</italic>
```

Result:

- all **text** nodes of the para element have been removed.
- mapped bold and italic text content remain
- However, bold and italic item **sequence** follow that of the **target** XML/schema file!

Target Driven (Standard) - properties
Standard mapping means the normal method of mapping used in MapForce, i.e.:

- Mixed content text node content is not supported/mapped.
- The sequence of child nodes is dependent on the target XML/schema file.

  In this example:
  For each **para** element, first **map all bold** items, then map **all italic** items. This results in the child item sequence shown above: bold, bold - italic, italic. The content of each item is mapped if a connector exists.

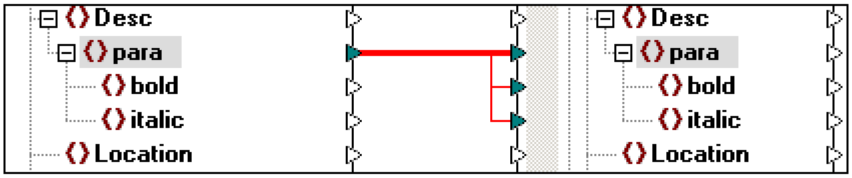
Please note:
If one of the child nodes/items use the **anyType** datatype, then the node **content** is not mapped - only the **empty** item/node name is transferred to the target component!

The **anyType** datatype, allows unconstrained content (including mixed content), and MapForce cannot automatically ascertain the varied structure of such types of nodes.

Change the datatype to anySimple type, or a more specific type e.g. xs:string, if empty nodes appear in the output window, or define a complexType in the schema and map the respective items.

Copy-all mapping:

1. Right click the para connector and select **Copy-all** from the popup window.
   The connector now appears as a solid line with the child items branching out of, and below it. Please see "Copy-all connections" for more information.



Chapter 5

Copy-all connections

5 Copy-all connections

This type of connection allow you to organize your workspace and automatically connect **all** identical items in source and target components, meaning that, depending on the source and target **type**:

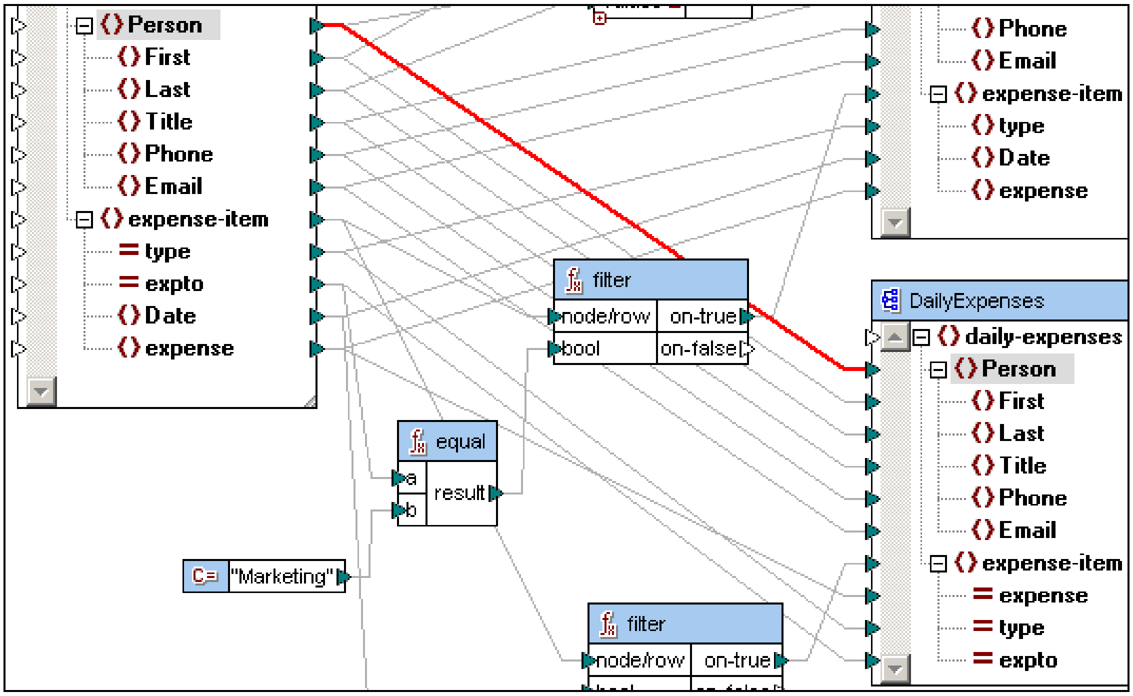
- all source child items are **copied** to the target component, if either the source and target **types** are **identical**, or if the target type is xs:anyType
- if the source and target **types** are **not identical,** and if the target type is not xs:anyType, the source data is transferred/mapped to the respective target items of the same name and the same hierarchy level. If the names of the target items differ, then the target item is not created.
- Note that only the names of the child items, but not their individual types, are compared/matched.

Currently Copy-all connections are supported:

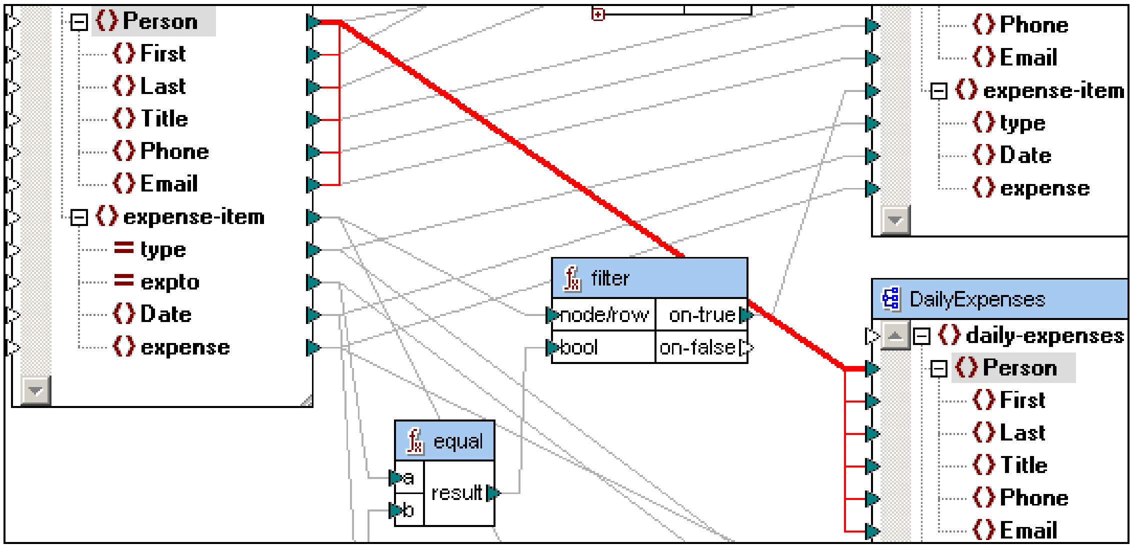
- between XML schema complex types, and
- between complex components (XML schema, database) and complex user-defined functions/components containing the same corresponding complex parameters, please see "Complex output components - defining" for an example.

The example below shows these connectors using the **MarketingAndDailyexpenses.mfd** file in the ...\MapForceExamples folder.

1. Right click the Person component and select "Copy-all" from the context menu. A prompt appears reminding you that the target connectors will be deleted.



2. Click OK if you want to create Copy-all connectors.



All connectors to the target component, and all source and target items with identical names are created.

Please note:

- When the existing target connections are deleted, connectors from other source components, or other functions are also deleted.
- This type of connection cannot be created between an item and the root element of a schema component.
- Individual connectors cannot be deleted, or reconnected from the Copy-all group, once you have used this method.

Copy-all connections and user-defined functions
When creating Copy-all connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

Chapter 6

MapForce How To...

6 MapForce How To...

This section deals with common tasks that will be encountered when creating your own mappings.

The tasks covered are:

- Mapping multiple tables to one XML file
- How to map data to the root element of target components
- Using boolean values in XSLT 1.0
- Mixed content data and MapForce
- Defining the Priority context
- MapForce command-line parameters
- Using input functions to override values, and act as parameters in command line execution
- Filter components - Tips
- Node testing

6.1 Mapping multiple tables to one XML file

Mapping multiple hierarchical tables to one XML output file

- You have a database and want to extract/map a certain number of tables into an XML file.
- Primary and foreign-key relationships exist between the tables
- Related tables are to appear as child elements in the resulting XML file.

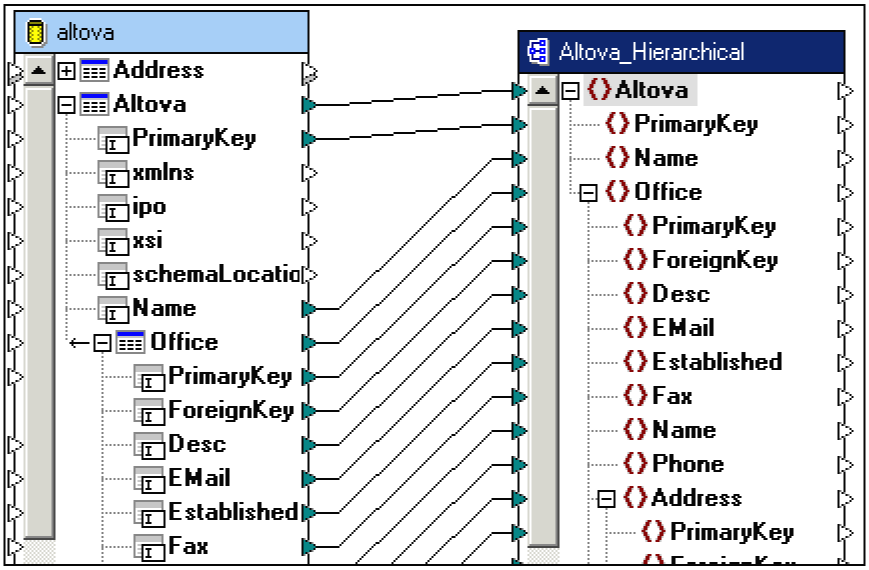
The "**DB_Altova_Hierachical.mfd**" sample file in the **...\MapForceExamples** folder shows how this can be achieved when mapping from an hierarchical database. The Altova_Hierarchical.xsd schema is also supplied in the same folder. The schema structure is practically identical to the Access database hierarchy. (The same method can also be used to map flat format XML/SQL databases.)

The MS Access database, **Altova.mdb**, is supplied in the **...\MapForceExamples\Tutorial** folder.

Schema prerequisites:

- All tables related to Altova, appear as child items of the target root element.
- To preserve the table relationships all mappings have been created under the Altova table in the database component.

  The diagram below shows the mapping of the hierarchical Access database to Altova_Hierarchical.xsd.



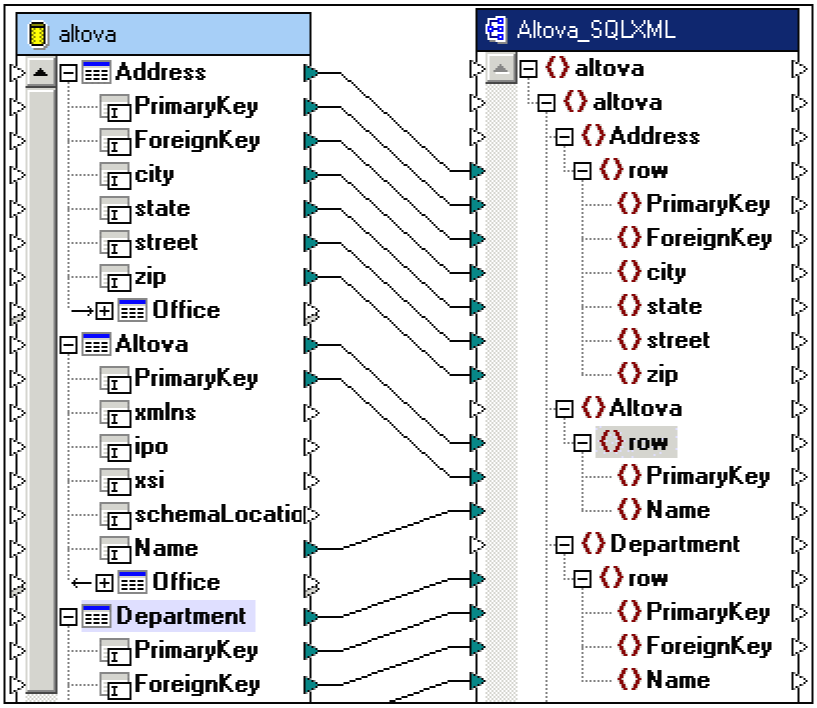
Mapping multiple flat file tables to one XML output file

The following diagram shows the same type of mapping to a flat file SQL/XML database schema.

Schema prerequisites:

- The **schema** structure has to follow the SQL/XML specifications.
- XMLSpy has the ability to create such an SQL/XML conformant file from an SQL database, by using the menu option **Convert | Create Database Schema**. You can then use the **schema** as the target in MapForce.
- In this case each table name is mapped to the row child element, of the same element

name in the schema, i.e. Address is mapped to the **row** child element of the **Address** element



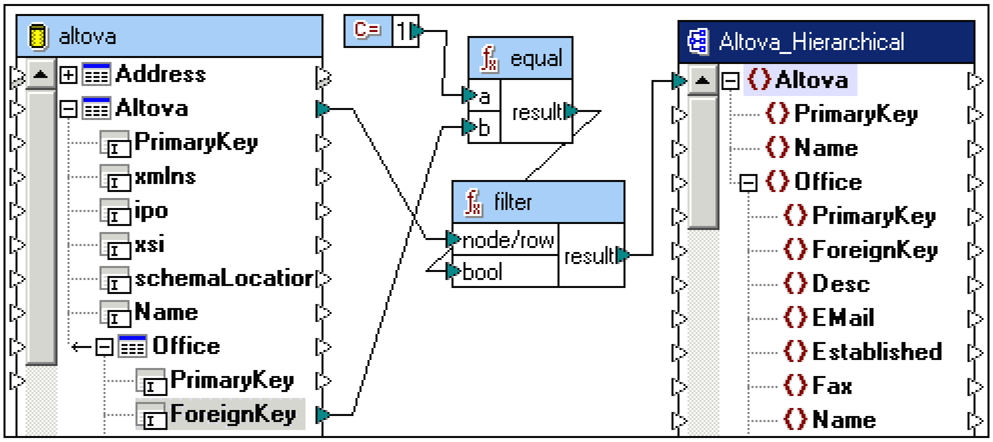
- Please note that the above example **DB_Altova_SQLXML.mfd**, does not preserve the table relationships, as mappings are created from several different "root" tables.

6.2 Mappings and root element of target documents

Root element of target XML files
When creating a mapping to the root element of the target Schema/XML file, please make sure that only one element, or record, is passed on to the target XML, as an XML document may only have one root element.

Use the filter component to limit the mapped data to a single element or record.

- In the example below, the ForeignKey is checked to see if it is 1, and only then is one Altova element passed on to the target root element.
- If no mappings exist from any of the source items to the target root element, then the root element of the target schema is inserted automatically.



Root element not limited:
If you do not limit the target schema root element, then all source elements/records are inserted between the first root element.

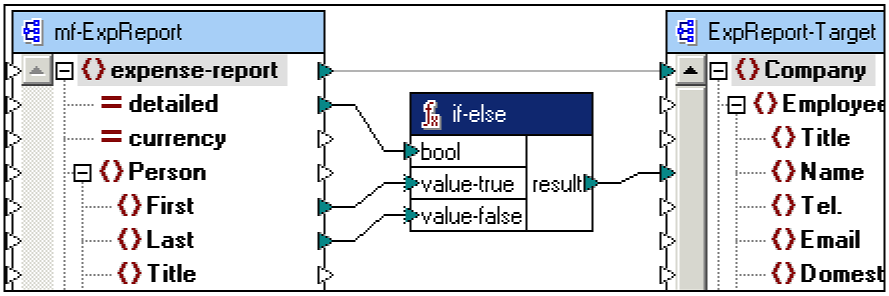
6.3 Boolean values and XSLT 1.0

Currently XSLT processors can only process values as strings. The values supplied by the "detailed" element in this example, can only be "true" or "false" (as defined in the schema file).

The example below tries to create an **if-else construct**, using the bool value of "detailed". Depending on the content, you should either see the First, or Last name of the Person element in the Target schema.

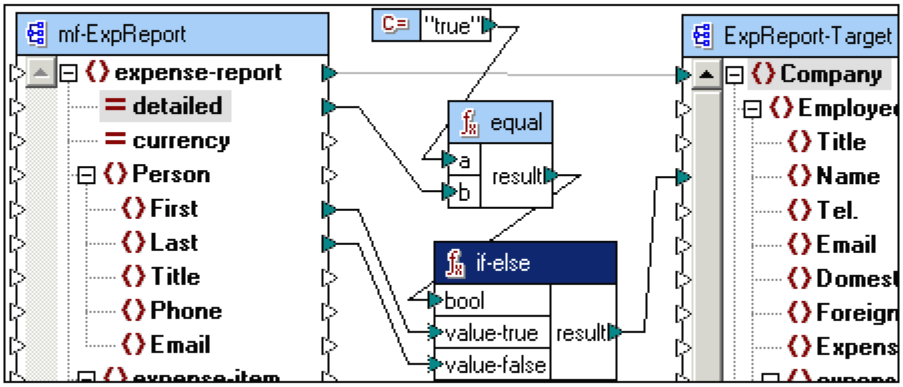
Trying out this mapping however, shows that whatever the bool value of detailed is, true or false, you will always have the contents of First in the target schema. XSLT currently takes **all string input** as True, so this method cannot be used to directly check a boolean value.

Clicking the "Insert Condition" icon inserts the IF-Else condition function.



To use boolean values as comparison values in XSLT:

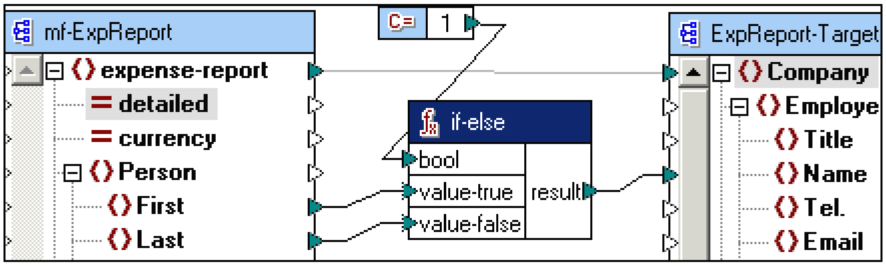
1. Supply a boolean value using the **constant** component, e.g. true.
2. Use the **equal** component to check if the value of the constant, is equal to the content of the boolean node, detailed.
3. Pass the **result** of the comparison on to the **bool** parameter of the **if-else** condition. If the **detailed** element supplies **true**, then the equal result parameter is also true.



- If the bool value (of if-else) is **true**, then the value of **First** is passed on to the target schema.
- If **false**, then the value of **Last** is passed on to the target schema.

Forcing boolean values:

There might be instances where you want to predefine, or force the result of a condition.



1. Connect the constant component directly to the **bool** parameter of an if-else/filter component.
2. Select the **Number** radio button in the "Insert constant" dialog box, and
3. Enter **1** for True, and **0** for false - depending on the condition you want satisfied.

6.4 Boolean comparison of input nodes

Data type handling in boolean functions (difference between MapForce 2006 SP1 and SP2)
During the evaluation of the core functions, less-than, greater-than, equal, not-equal, less equal, and greater equal, the evaluation result of two input nodes depends on the input values as well as the data types used for the comparison.

Example:
The 'less than' comparison of the integer values 4 and 12, yields the boolean value "true", since 4 is less than 12. If the two input strings contain '4' and '12', the lexical analysis results in the output value false", since '4' is alphabetically greater than the first character '1' of the second operand (12).

If all "input" data types are of the same type, e.g. all input nodes are numerical types, or strings, then there is no difference between the SP1 and SP2 versions.

Differing input node types (only version SP2):
If the input nodes are of differing types, e. g. integer and string, or string and date, then version SP2 introduces a new rule:

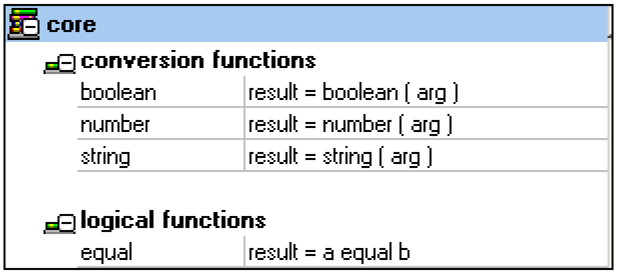
The data type used for the comparison **is always the most general, i. e. least restrictive, input data type** of the two input types.

Before comparing two values, all input values are converted to a common datatype. Using the previous example; the datatype "string" is less restrictive than "integer". Comparing integer value 4 with the string '12', converts integer value 4 to the string '4', which is then compared with the string '12'.

The type handling for comparing mixed types, follows the XSLT2 guidelines and prevents any content-sensitive type conversion strategies. The advantage is that the logic is fixed by the mapping and does not change dynamically.

Additional checks:
Version SP2 additionally checks mappings for incompatible combinations and raises validation errors and warnings if necessary. Examples are the comparison of dates with booleans, or "datetimes" with numerical values.

In order to support explicit data type conversion, Version SP2 introduces three new **type conversion** functions to the core library: "boolean", "number" and "string". In the previously mentioned context, these three functions are suitable to govern the interpretation of comparisons.



core

conversion functions

boolean	result = boolean (arg)
number	result = number (arg)
string	result = string (arg)

logical functions

equal	result = a equal b

Adding these conversion functions to input nodes of related functions might change the common data type and the result of the evaluation in the desired manner. E. g. if string nodes store only numeric values, a numerical comparison is achieved by adding the "number" conversion function (in the **conversion** section of the **core** library) to each input node.

6.5 Priority Context

When applying a function to different items in a schema or database, MapForce needs to know what the context node will be. All other items are then processed relative to this one. This is achieved by designating the item (or node) as the priority context.

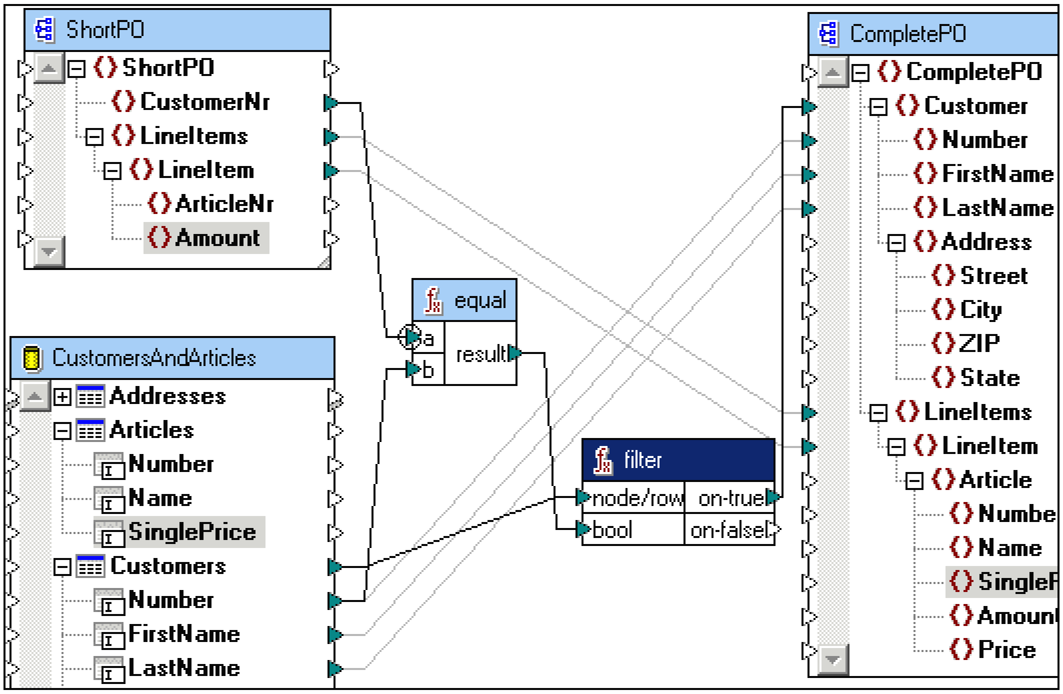
A simplified version of the complete **DB-CompletePO.mfd** file available in the **...\MapForceExamples** folder, is shown below.

Please note that there are multiple source components in this example. **ShortPO** is a Schema with an associated XML instance file, while **CustomersAndArticles** is a database schema. The data from both, are then mapped to the CompletePO schema / XML file. The priority context icon, is enclosed in a circle as a visual indication.

- The **CustomerNr** in ShortPO is compared with the item **Number** in the database.
- **CustomerNr** has been designated as the **priority context**, and is placed in the **a** parameter of the equal function.
- The **CustomersAndArticles** database is then searched (**once**) for the **same** number. The **b** parameter contains the Number item from the database.
- If the number is found, then the result is passed to the **bool** parameter of the **filter** function.
- The **node/row** parameter passes on the **Customer** data to "on-true" when the bool parameter is true, i.e. when the same number has been found.
- The rest of the customer data is then passed on as: Number, FirstName, LastName items, are all connected to the corresponding items in the target schema.

Designating the **b** parameter of the equal function (i.e. item Number), as the **priority context** would cause:

- MapForce to load the first Number into the **b** parameter
- Check against the **CustomerNr** in **a,** if not equal
- Load the next Number into **b**, check against a, and
- Iterate through every Number in the database while trying to find that number in ShortPO.



Priority context and user-defined functions:
If a user-defined function has been defined of type "inline", the default setting, then a priority context cannot be defined on one of the parameters of the user-defined function. The user-defined function can, of course, contain other "Standard" user-defined functions which have priority contexts set on their parameters.

If the user-defined function was originally of type "standard" with a priority context, and was subsequently changed to one of type "inline", then the priority context is hidden and deactivated. Changing the same function back to "standard", shows the priority context and enables it once again.

Please see

6.6 Command line parameters

The command line parameter syntax for MapForce is shown below.

Please note that MapForce currently supports XERCES version 2.2.0.

General syntax:
MapForce.exe Filename [(/BUILTIN | /XSLT | /XSLT2 | /XQuery | /Java | /CS | /CS:(VS2005|VS2003|VS2002|BORLAND|MONO) | /CPP | /CPP*:(VC8|VC71), (MSXML|XERCES),(LIB|DLL),(MFC|NoMFC)*) outputdir [/LOG logFileName]]

- The square brackets **[...]** denote optional.
- The round brackets **(...)** denote a parameter group containing several choices.
- The **pipe** symbol **|** denotes OR, e.g. /XSLT or /Java

Description of parameters:

Filename	path and YourMAPFORCEfile.MFD **If the path, or file name contains a space, please use quotes around the path/file name i.e. "c:\Program Files\...\Filename"**
/BUILTIN	generates all outputs using the built-in transformation engine
/XSLT	generates all XSLT files
/XSLT2	generates XSLT files
/XQuery	generates XQuery code
/Java	generates the Java application
/CS	generates the C# application using the configuration of the mapping settings
/CS: ...	generates the C# application using special configuration given in option-field of the command-line parameters
VS2005	generates Microsoft VisualStudio.Net 2005 solution files
VS2003	generates Microsoft VisualStudio.Net 2003 solution files
VS2002	generates Microsoft VisualStudio.Net (2002) solution files
BORLAND	generates Borland C#Builder 1.0 project-group-files
MONO	generates makefile for MONO environment
/CPP	generates the C++ application using the configuration of the mapping-settings
/CPP: ...	generates the C++ application using special configuration given in options-field of the command-line parameters
VC8	generates Microsoft VisualStudio 2005 solution files
VC71	generates Microsoft VisualStudio.Net 2003 solution file
MSXML	generates code using MSXML 4.0
XERCES	generates code using XERCES
LIB	generates code for static libraries
DLL	generates code for dynamic-linked-libraries
MFC	generates code supporting MFC
NoMFC	generates code without MFC support
Builtin	generates code using the built in transformation engine
Outputdir	directory the log file is to be placed in
/LOG	LogFileName, name of the log file to be generated

Please Note:
VC6 workspace files are always generated

Examples:

MapForce.exe Filename starts MapForce and opens the file defined by Filename.

I) generate all XSLT files and output a log file.

MapForce.exe Filename /XSLT outputdir /LOG logFileName

II) generate a Java application and output a log file.

MapForce.exe Filename /Java outputdir /LOG logFileName

III) generate a C# application and output a log file.

MapForce.exe Filename /CS outputdir /LOG logFileName

IV) generate a C++ application using the configuration of the mapping settings, and output a log file.

MapForce.exe Filename /CPP outputdir /LOG logFileName

V) generate a C++ application using the /CPP switch, restricting your C++ compiler options.

MapForce.exe Filename /CPP:(MSXML|XERCES),(LIB|DLL),(MFC|NoMFC|Builtin) outputdir [/LOG logFileName]

MapForce.exe Filename /CPP:MSXML,LIB,MFC

Generates the C++ application using all of the first choices, in this example:

- compile for C++
- use MSXML4.0
- generate code for static libraries
- have generated code support MFC

MapForce.exe Filename /CPP:XERCES,DLL,NoMFC outputdir /LOG logFileName

Generates the C++ application using all of the second choices, in this example:

- compile for C++
- use XERCES
- generate code for dynamic libraries
- generated code not to support MFC
- create a log file in the outputdir with the name LogFileName

VI) generate all output files (target XML document, and databases) using the built-in transformation engine.

MapForce.exe Filename outputdir /BUILTIN

6.7 Input values, overrides and command line parameters

MapForce allows you to create special input functions that can:

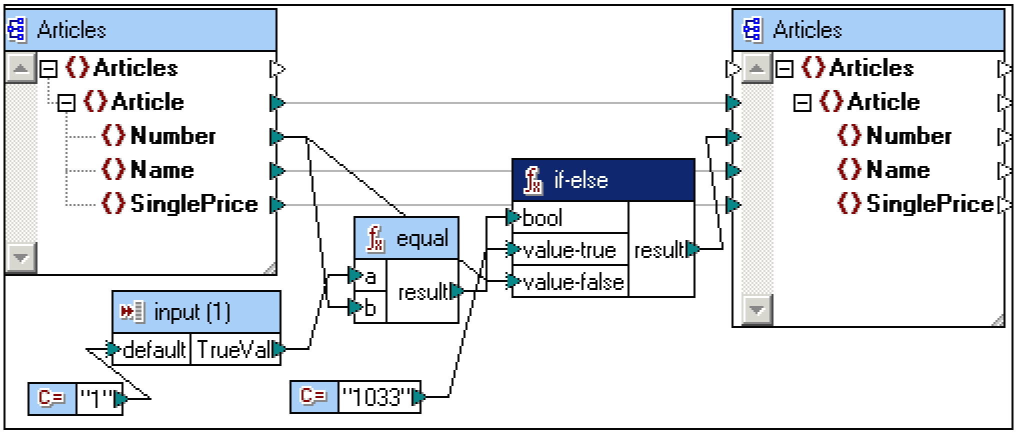
- define an **override,** or alternative, value for data being input by the current mapping, and
- use this input component as a **parameter** in the command line execution of the compiled mapping.

Please note:
This specific type of input function cannot be used **inside** a user-defined function.

The mapping below, uses such an input function. The aim of this mapping is to search for a specific article number, and replace it with a value 1033, if found. If the search is not successful, retain the current number.

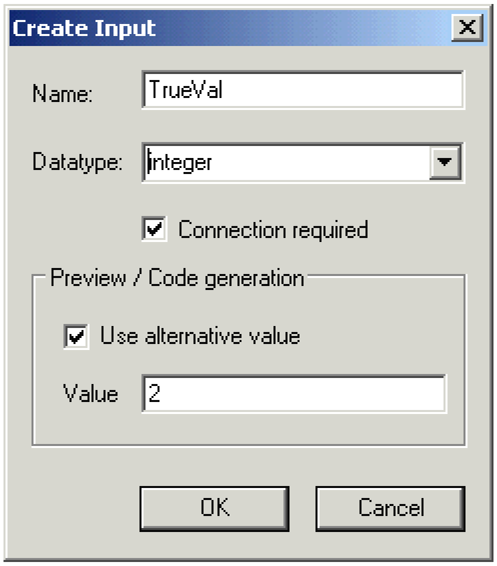
What the input function allows you to do, is override the current input which is 1, and replace it with whatever you define in the input function. Please note that the input in this example is a constant, i.e. 1, but that this will generally not be the case in a complex mappings, where the input can be any type of data from any input source.

The input function further doubles as an **input parameter** for the command line execution of the generated mapping code!



The above example uses the Articles.xsd schema and Articles.xml files, available in the ...\MapForceExamples folder. The article numbers in the source XML file are 1, 2, 3, and 4.

1. Use the menu option **Function | Insert Input** to insert the component.
   This opens the Create Input dialog box.
2. Enter a name for the function and select the datatype you want to use.
3. Click in the **Value** field and enter a value. In this case, enter a value different from the one supplied by the constant e.g. 2.



Name: TrueVal

Datatype: integer

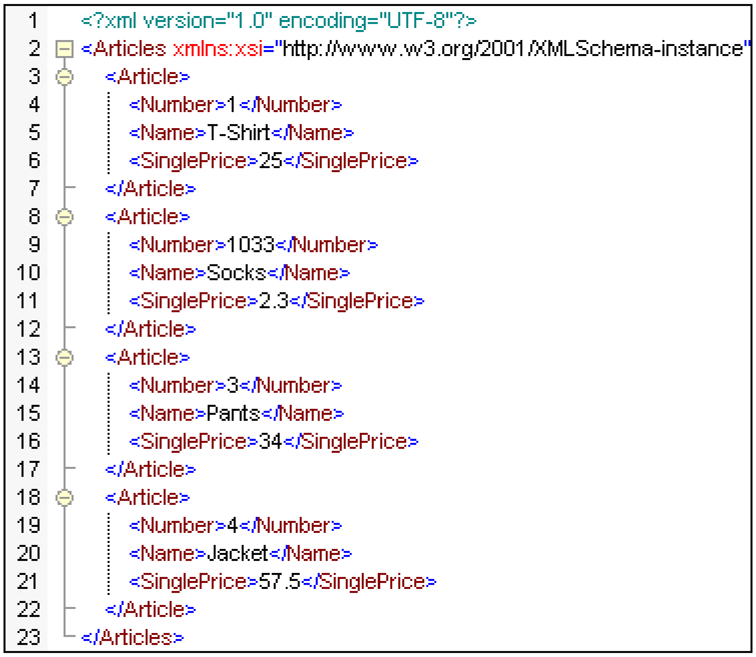
☑ Connection required

Preview / Code generation

☑ Use alternative value

Value 2

4. Click the Output tab to see the result of the mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Article>
    <Number>1</Number>
    <Name>T-Shirt</Name>
    <SinglePrice>25</SinglePrice>
  </Article>
  <Article>
    <Number>1033</Number>
    <Name>Socks</Name>
    <SinglePrice>2.3</SinglePrice>
  </Article>
  <Article>
    <Number>3</Number>
    <Name>Pants</Name>
    <SinglePrice>34</SinglePrice>
  </Article>
  <Article>
    <Number>4</Number>
    <Name>Jacket</Name>
    <SinglePrice>57.5</SinglePrice>
  </Article>
</Articles>
```

The original article number 2, has been changed to 1033. The value supplied by the input function has taken precedence over the value supplied by the constant.

Input values and Code generation:
Values or strings entered in the "Create/Edit input" dialog box are only applicable when:

- Previewing results in the Output tab, or
- when generating program code in XSLT 1.0/2.0! The data directly supplied by the input icon are used when generating code.

Using input values as parameters in command line execution of mappings:
Input values can be used as parameters when calling the generated mapping, where:

- the generated application name is **Mapping.exe**
- the input value name "**TrueVal**" is the first parameter, and
- the input value "**2**" is the second parameter.

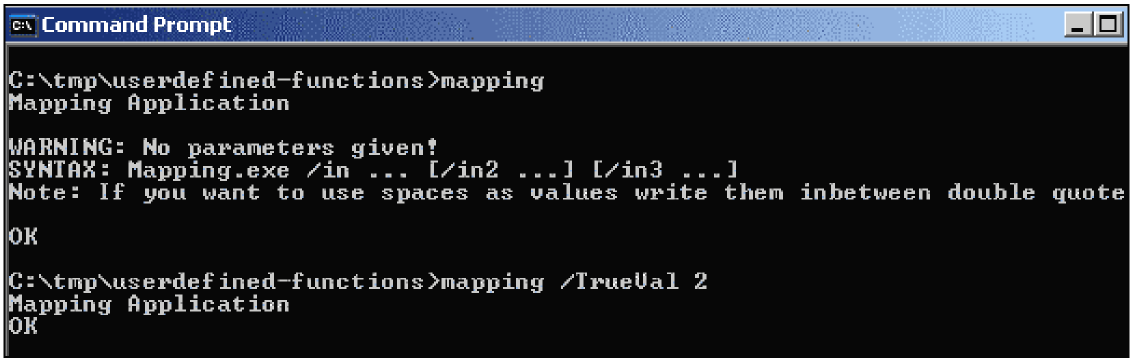
The command line thus becomes:

mapping.exe /TrueVal 2

Please note:

Running mapping.exe without parameters, displays a warning message, and help on the command line syntax needed.

- Alternative values are NOT used if the specific command line parameters are not supplied during command line execution of the generated EXE file, e.g. **mapping.exe.** In this case the default, or data supplied by the connected item is used.

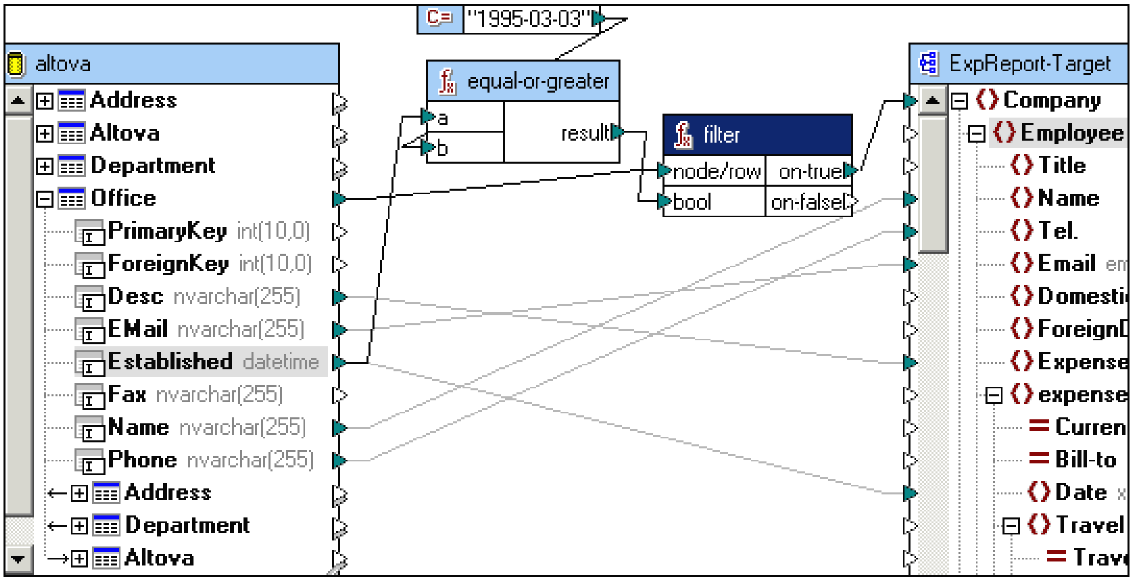


```
C:\tmp\userdefined-functions>mapping
Mapping Application

WARNING: No parameters given!
SYNTAX: Mapping.exe /in ... [/in2 ...] [/in3 ...]
Note: If you want to use spaces as values write them inbetween double quote

OK

C:\tmp\userdefined-functions>mapping /TrueVal 2
Mapping Application
OK
```

6.8 Filtering database data by date

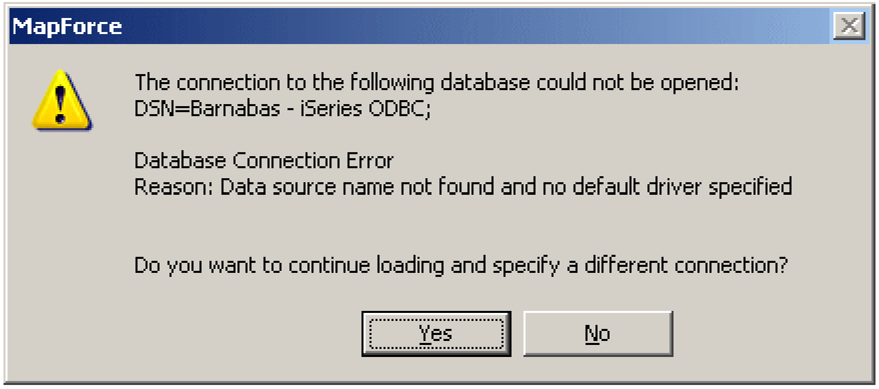
The example below shows how you can use the filter component to filter out database records according to a specific date.

- The Established field is defined as a Date/Time field in the database.
- The comparison date is entered into a Constant component, and is of type string.
- If the date record is greater than 1995-03-03, only then are the respective Office data passed on to the target file by the filter component.



6.9 Specifying alternate database resources

When opening a mapping file (*.mfd) containing a database component, it is possible that the database location has changed, or the data source name does not exist any more. If this is the case, a message box opens allowing you to continue the process and select a different database connection.



The connection to the following database could not be opened:
DSN=Barnabas - iSeries ODBC;

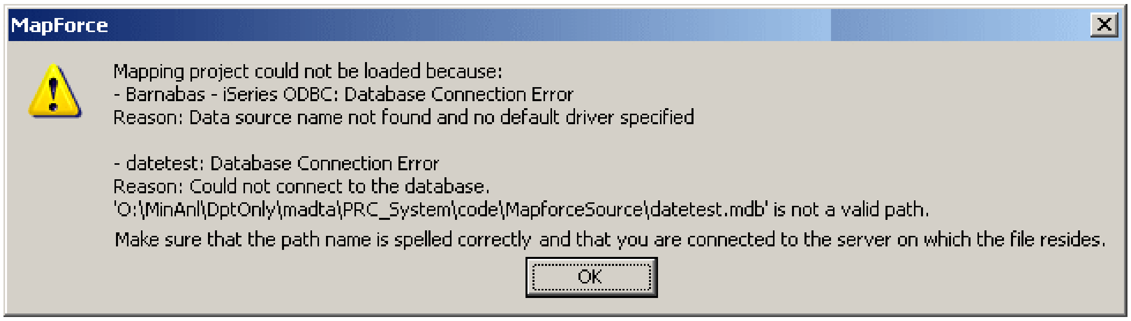
Database Connection Error
Reason: Data source name not found and no default driver specified

Do you want to continue loading and specify a different connection?

- Clicking No, halts the file loading process. Any additional database components/connections in the mapping are ignored.
- Clicking Yes, opens the "Select a source database" dialog box in which you can start the database selection process.

  If the database connection can be established, then the file opening process continues. If additional database connections cannot be opened, you are also prompted to select them anew.

  If the database connection cannot be established, then the loading process is halted. This is the same as clicking the No button in the dialog box. A message box is opened showing all errors that were encountered while trying to open the mapping file. The aim is to give you more information on why the mapping could not be loaded.



Mapping project could not be loaded because:
- Barnabas - iSeries ODBC: Database Connection Error
Reason: Data source name not found and no default driver specified

- datetest: Database Connection Error
Reason: Could not connect to the database.
'O:\MinAnl\DptOnly\madta\PRC_System\code\MapforceSource\datetest.mdb' is not a valid path.
Make sure that the path name is spelled correctly and that you are connected to the server on which the file resides.

The message box shows that two separate database connection errors occurred:

- The first database "Barnabas", could not be opened because the data source name was missing and a default driver was not specified.
- The second database "datetest" could not be opened because of an invalid path.

6.10 Filter components - Tips

The "filter" component is very important when querying database data, as it allows you to work on large amounts of data efficiently. When working with database tables containing thousands of rows, filters reduce table access and efficiently structure the way data is extracted. The way filters are used, directly affects the speed of the mapping generation.

This section will deal with methods enabling you to optimize data access and generally speed up the mapping process.

In general, use as few filter components as possible, and:

1. Avoid concatenating filter-components
2. Connect the "on-true/on-false" parameters, to parent items if possible, instead of child items directly
3. Connect the "on-false" parameter to map the complement node set, delivered by the on-true parameter
4. Don't use filters to map to child data, if the parent item is mapped
5. Use the "Priority context" to prioritize execution of unrelated items

Avoid concatenating filter components
Every filter-component leads to a loop through the source data, thus accessing the source **n** times. When you concatenate two filters, it loops **n*n** times.

Solution:
Use "**logical-and**" components to combine the boolean expressions of two filter-components. The result is a single filter component looping only **n**-times.

Connect the "on-true/on-false" parameter of the filter component, to target parent items
Filter components work best when they are connected to parent items containing child items, instead of individual items directly.

The filter **boolean** expression is therefore evaluated against the parent, **before** looping through the child elements. Using filters mapped from a database table will generate:

- "SELECT * FROM table WHERE <expression>" if the **parent** item is mapped, or
- "SELECT * FROM table", and then evaluate for each row, if child items are mapped

Please note:
> when connecting a filter from a source parent item, its also necessary to connect the on-true/on-false parameter to the parent target element. If this cannot be done, then do not apply this rule.

Connect the "on-false" parameter to map the complement node set
Connecting this parameter allows you quick access to the complement node set defined by the current mapping. The same tips apply when using this parameter, connect to parent items etc.

Don't use filters to map to child data, if the parent item is mapped
Using a filter to map data from a source parent to a target parent, automatically applies the **same filter** to **every child** item of the particular parent.

Filter components do not have to be used to supply filtered data to child items, if the parent item can be mapped! You can therefore map child data directly.

Use priority-context to prioritize execution when mapping unrelated items
Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context, MapForce does not know which loop needs to be executed first, it therefore

automatically selects the first table, or data source.

Solution:
Decide which table, or source data is to be looped/searched first, and then set the priority context on the connector to that table. Please see Priority Context for a more concrete example.

To define a priority context:

- Right click an input icon and select "Priority Context" from the pop-up menu.
  If the option is not available, mapping the remaining input icons of that component will make it accessible.

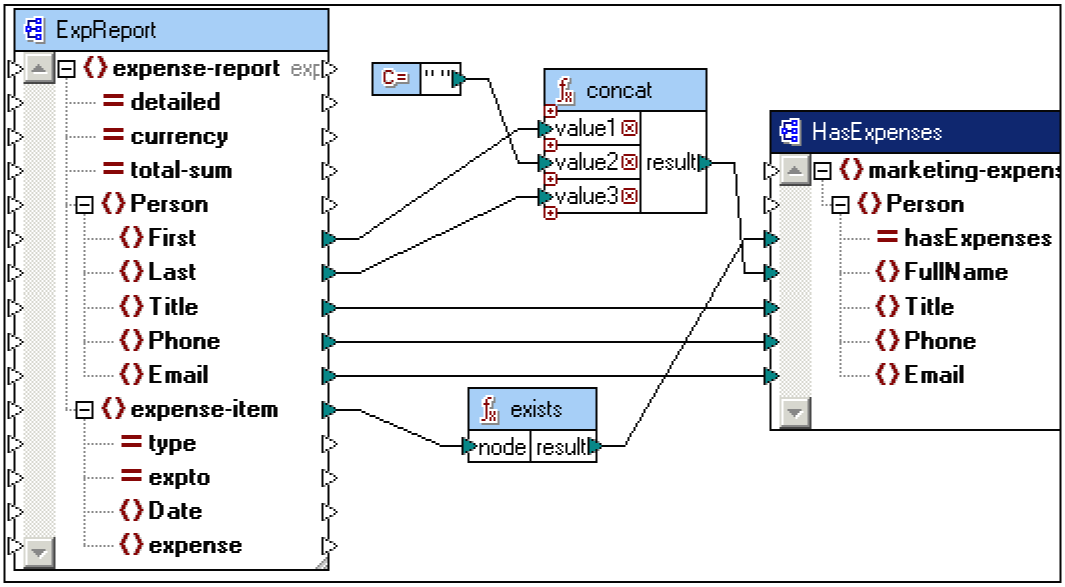
6.11 Node testing

The node testing functions allow you to test for the existence of nodes in the **XML instance** files. Elements or attributes defined as optional in the XML Schema, may, or may not, appear in the XML instance file. Use these functions to perform the specific node test and base further processing on the result.

Exists

Returns true if the node exists, else returns false.
The "HasMarketingExpenses.mfd" file in the ...\MapForceExamples folder contains the small example shown below.

If an expense-item exists in the source XML, then the "hasExpenses" attribute is set to "true" in the target XML/Schema file.

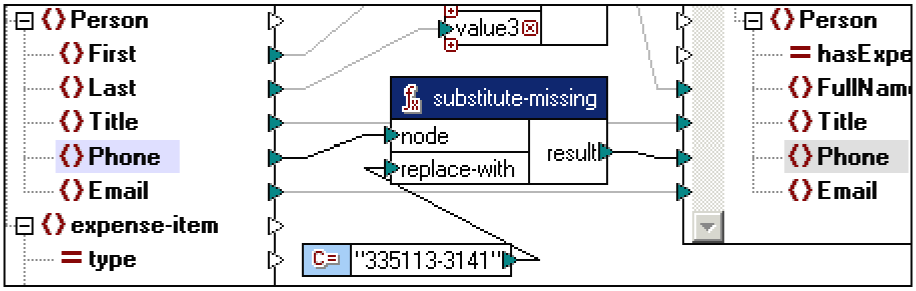


Not-exist

Returns false if the node exists, else returns true.

substitute missing

Used to map the current field content if the node exists in the XML source file, otherwise use the item mapped to the "replace-with" parameter.



In the image above, the existence of the node "Phone" is checked in the XML instance file. If the node is not present, then the value supplied by the constant is mapped.

6.12 Using DTDs as "schema" components

MapForce 2006 SP2 supports namespace-aware DTDs for source and target components. The namespace-URIs are extracted from the DTD "xmlns"-attribute declarations, to make mappings possible.

Adding DTD namespace URIs
There are however some DTDs, e.g. DTDs used by StyleVision, which contain xmlns*-attribute declarations, without namespace-URIs. These DTDs have to be extended to make them useable in MapForce.

- The DTD has to be altered by defining the xmlns-attribute with the namespace-URI as shown below:

```
<!ATTLIST fo:root
        xmlns:fo CDATA #FIXED 'http://www.w3.org/1999/XSL/Format'
        ...
>
```

Chapter 7

MapForce and Databases

7 MapForce and Databases

MapForce allows you to not only map database data to XML documents, but you can do the reverse as well, map XML data to databases and even create mappings between databases! MapForce takes primary and foreign key constraints into account and also generates transaction data which ensures data integrity.

Please note:

> **XQuery** code can currently only be generated for XML data sources! Database access currently requires that you use one of the programming languages: Java, C#, or C++.

Currently supported databases (and connection types) are:

- Microsoft Access (ADO); versions 2000 and 2003
- Microsoft SQL Server (ADO)
- Oracle (OCI)
- MySQL (ODBC)
- Sybase (ODBC)
- IBM DB2 (ODBC)

- Any ADO (compliant database)
- Any ODBC (compliant database)

7.1 JDBC driver setup

JDBC drivers have to be installed for you to compile Java code when mapping database data.

Overview

This section describes how to download and install JDBC drivers and how to use them with **Ant** and **JBuilder**. A **JBuilder** project file and **Ant** build scripts are generated by MapForce when generating Java code.

JDBC drivers are used by MapForce generated Java applications to connect to, and exchange data with several different databases. These JDBC drivers need to be installed first, to successfully run the generated Java application(s).

In general JDBC drivers can be found at http://industry.java.sun.com/products/jdbc/drivers

MapForce generated Java applications were tested with the following JDBC-drivers:

- MS Access
- MSSQL2000
- Oracle 9i
- MySQL
- Sybase
- IBM DB2

This section assumes the following:

- the reader is familiar with setting Java CLASSPATHs
- Java SDK and Ant, or JBuilder is already installed and is working correctly
- at least one of the databases described below is running and the minimum privilege - read-only, is granted

MS Access

The JDBC-ODBC-bridge is already installed with Java SDK.

Java internal usage

Driver	sun.jdbc.odbc.JdbcOdbcDriver
URL	jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);DBQ=Sourcename...

Microsoft SQL Server 2000

Download from http://www.microsoft.com/sql/

Ant Settings

Please make sure that the following jar file entries are in the CLASSPATH:

```
C:\Program Files\Microsoft SQL Server 2000 Driver for JDBC\lib\
msbase.jar;C:\Program Files\Microsoft SQL Server 2000 Driver for
JDBC\lib\mssqlserver.jar;C:\Program Files\Microsoft SQL Server 2000
Driver for JDBC\lib\msutil.jar
```

assuming that "`C:\Program Files\Microsoft SQL Server 2000 Driver for JDBC`" was your installation folder.

JBuilder Settings

Use the menu option **Tools | Configure JDKs...** then click **Add** to add all the jar files listed above.

Java internal usage

Driver	com.microsoft.jdbc.sqlserver.SQLServerDriver
URL	jdbc:microsoft:sqlserver://localhost

Oracle 9i

Download the Oracle9i Release 2 (9.2.0.3) driver for JDK 1.4: **ojdbc14.jar**
from http://otn.oracle.com/software/tech/java/sqlj_jdbc
You will need to have an account, or sign up to the Oracle Technology Network to access these drivers.

Ant Settings
Add the full path to ojdbc14.jar to the CLASSPATH.

JBuilder Settings
Use the menu option **Tools | Configure JDKs...** then click **Add** to add the jar file above.

Java internal usage

Driver	oracle.jdbc.OracleDriver
URL	jdbc:oracle:oci:@localhost

7.2 Development environments for code generation

Below is a list of the requirements for each of the development environments, as well as other tools, that are needed when generating code using MapForce.

Java Minimum requirements:
Java2 SDK SE (Standard-Edition) 1.4.1
Apache ANT 1.5.3

Other vendor supported IDEs:
Borland JBuilder 8

Optional:
Sun 1 Studio - import of ANT build-file into IDE

C# Minimum requirements:
Microsoft .Net Framework SDK 1.0 - for compilation and build process

Additionally:
Microsoft Visual Studio.NET 2002 / 2003
Microsoft Visual Studio.NET 2005
Borland C#Builder 1.0
MONO 0.26
Optional:

C++ Minimum requirements:
Microsoft Visual Studio 6.0 - for compilation, build process and as IDE.

7.3 Mapping XML data to databases

MapForce allows you to not only map database data to XML documents, but you can do the reverse as well, map XML data to databases and even create mappings between databases! MapForce takes primary and foreign key constraints into account and also generates transaction data which ensure data integrity.

Database functions (table actions) currently supported by MapForce:

- Insert
- Update
- Delete
- Database key field handling

Examples for each of these table actions follow, and are of a simple nature to get you acquainted with how to achieve the specific goals.

Currently supported databases (and connection types) are:

- Microsoft Access (ADO)
- Microsoft SQL Server (ADO)
- Oracle (OCI)
- MySQL (ODBC)
- Sybase (ODBC)
- IBM DB2 (ODBC)

- Any ADO (compliant database)
- Any ODBC (compliant database)

Files used in this section:

Altova_Hierarchical.xsd	the hierarchical schema file, containing identity constraints
Altova-cmpy.xml	the Altova company data file which supplies the XML data
Altova.mdb	the Altova MS-Access database file, which functions as the target database

All these example files are available in the ...\MapForceExamples folder

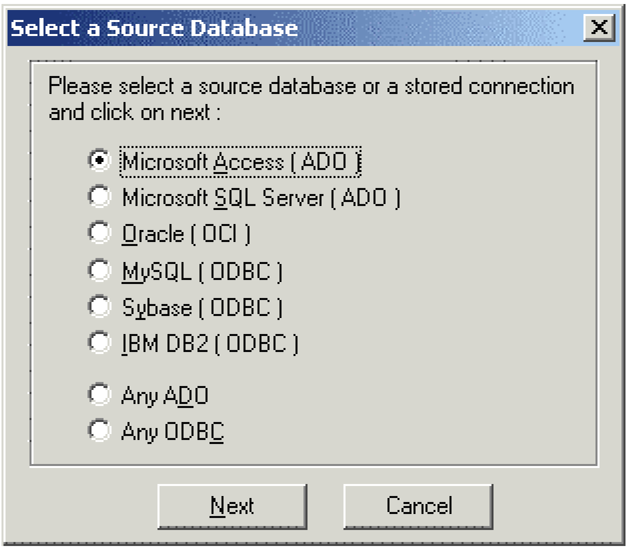
Please note:

This section makes heavy use of the **Altova.mdb** database, to show the database-as-target functionality of MapForce. Make sure you backup the file before you try any of the examples shown here.

7.3.1 Setup of XML to database mapping

Setting up an XML to database mapping, is in no way different from the methods previously described.

1. Click the **Insert Schema | XML instance** icon, and select the **Altova_Hierarchical.xsd** .
2. Select the **Altova-cmpy.xml** file as the XML instance file. Click the **Altova** entry, and hit the ***** key on the numeric keypad to view the items; resize the component if necessary.
3. Click the **Insert Database** icon, select the Microsoft Access (ADO) entry and click Next.
4. Click the **C++** icon in the title bar to specify the language the generated code should support. This setting also loads the language related library into the Libraries window.



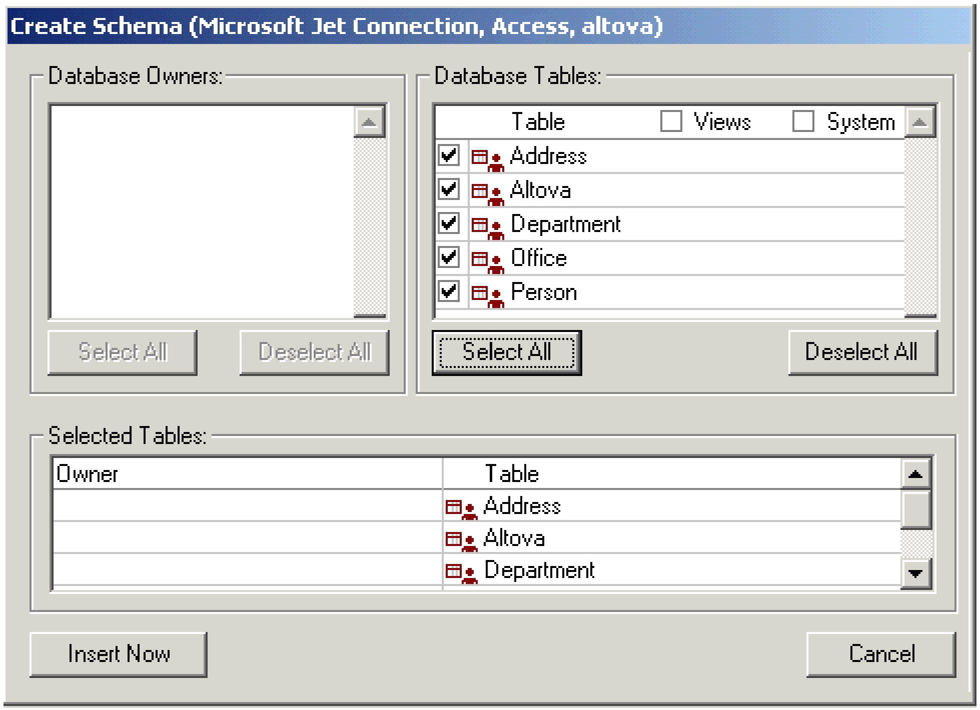
Select a Source Database

Please select a source database or a stored connection and click on next :

● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (OCI)
○ MySQL (ODBC)
○ Sybase (ODBC)
○ IBM DB2 (ODBC)

○ Any ADO
○ Any ODBC

5. Click the Browse button to select the **altova.mdb** database available from the **...\MapForceExamples\Tutorial** folder, and click Next.
This dialog box allows you to define the specific Tables, Views or System tables that you want to appear in the Database component.


Create Schema (Microsoft Jet Connection, Access, altova)

Database Owners:

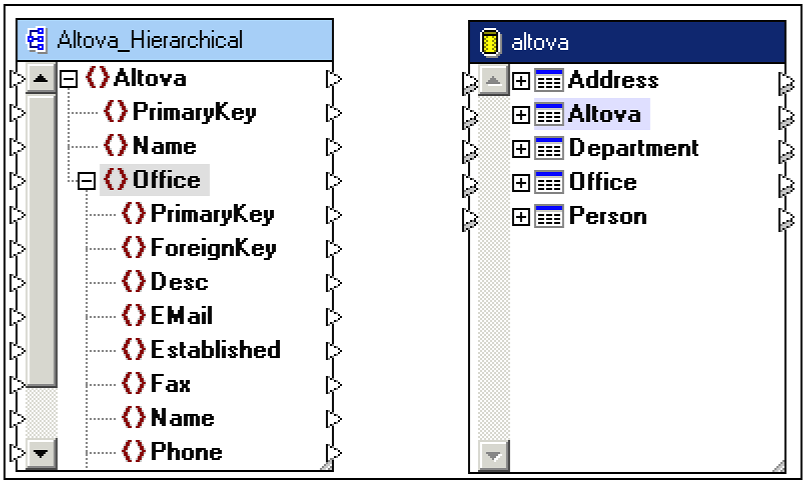
Database Tables:

Table	□ Views	□ System
☑ Address		
☑ Altova		
☑ Department		
☑ Office		
☑ Person		

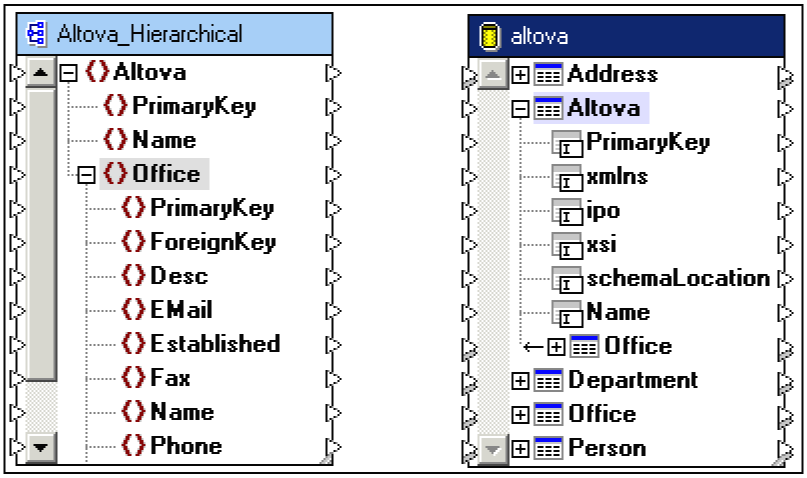
Selected Tables:

Owner	Table
	Address
	Altova
	Department

6. Click the **Select All** and then the **Insert Now** buttons, to insert the database.



7. Click the + expand icon of the **Altova** item, to display the Altova table fields.



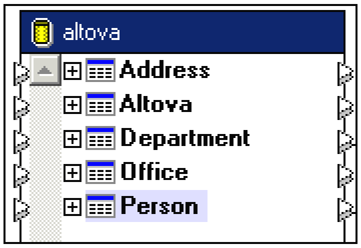
Please note:

Creating mappings between database components is not possible if you select XSLT, XSLT2, or XQuery as the target language. XSLT does not support database queries.

Once you have defined the database settings using the method described above, they cannot be changed by editing the component settings of the database component (right click and select Component settings), please see the Reference | Component Settings for more information.

7.3.2 Components and table relationships

Table relationships are easily recognized in the database component. The database component displays each table of a database, as a "**root**" table with all other related tables beneath it in a tree view.



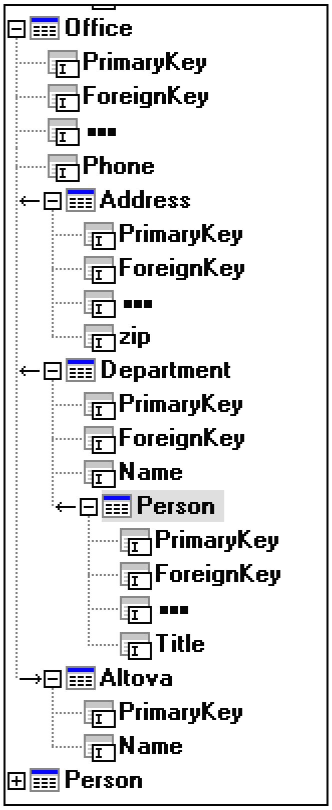
Let us call the table names visible in the above diagram "**root**" tables, i.e. they are the top level, or **root** of the tree view. Expanding a table name displays all the tables related to it. The "root" tables are usually displayed in alphabetical sort order; this has no bearing on the actual table relationships however.

When creating queries/mappings of databases with relations, including flat format SQL/XML databases, make sure that you create mappings **between tables** that appear under **one of the "root" tables**, if you want the table relationships to be maintained i.e. when creating queries that make use of joins.

The graphic below, shows the expanded **Office** "root" table of the Altova database. The arrows to the left of the expand/contract icons of each table name, as well as the indentation lines, show the table relationships.

Starting from the **Office** table and going down the tree view:

- **Arrow left,** denotes a child table of the table above, **Address** is a child table of Office.
- Department is also a child of Office, as well as a "sibling" table of Address, both have the same indentation line. Person is also a child table of Department.
- **Arrow right,** denotes a parent of the table above, **Altova** is the parent of the Office table.



Which "root" tables should I use when I am mapping data?
When creating mappings to database tables, make sure you create mappings using the **specific** "root" table as the top level table.

E.g.
suppose you only want to insert or update Person table data. You should then create mappings using the Person table as the "root" table, and create mappings between the source and target items of the Person fields you want to update.

If you want to update Department and Person data, while retaining database relationships between them, use the Department table as the "root" table, and create mappings between the source and target items of both tables.

7.3.3 Database action: Insert

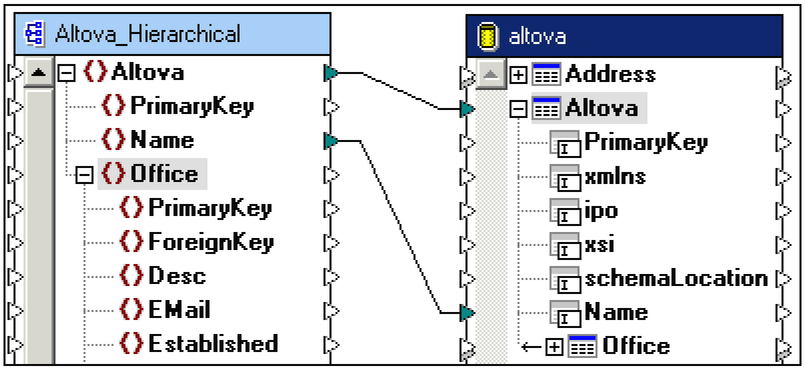
The first example in this section, deals with the simple task of **adding** a new office orgchart to the Altova table. The only fields available in the Altova table are: PrimaryKey and Name.

The second example **inserts** related office tables to the new orgchart record.

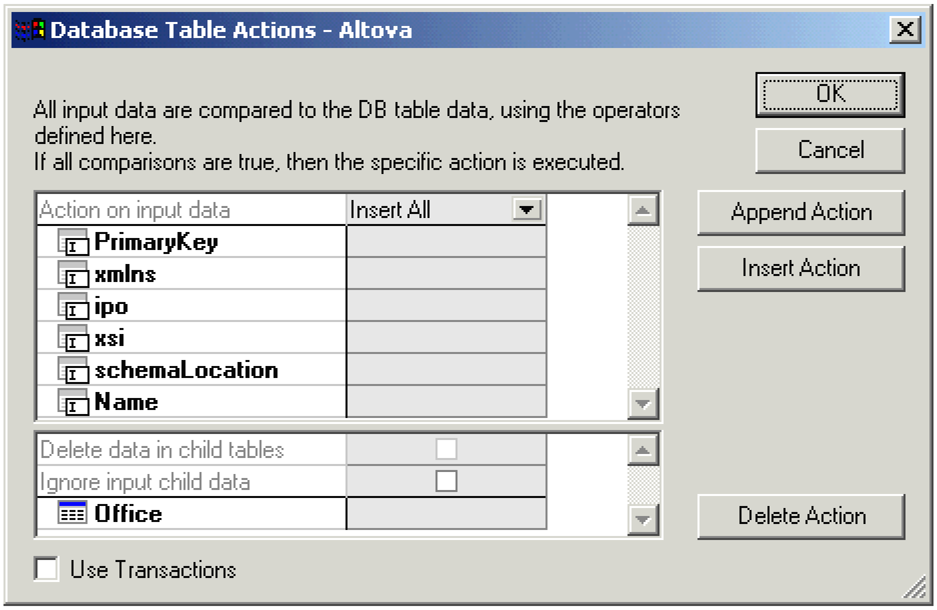
1. Insert the Altova_Hierarchical.xsd schema (and assign **altova-cmpy.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.
3. Create the following mappings:

   Altova to Altova, and
   Name to Name

   Please note: If all Altova, Office etc. items are automatically mapped, the option "Auto-connect children" is active. Select undo, and then the menu option **Connection | Auto-connect matching children,** to disable this option.



4. Right click the **Altova** entry and select the menu item "**Database Table Actions**". There is currently only one table action column defined in this dialog box, **Insert All**. (Update if... and Delete if... table actions are selected by clicking the column header combo box, whereas **additional** table actions, can be defined by clicking the Append, or Insert Action buttons.)



All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on input data	Insert All
PrimaryKey	
xmlns	
ipo	
xsi	
schemaLocation	
Name	

Delete data in child tables	☐
Ignore input child data	☐
Office	

☐ Use Transactions

The table action, Insert All, inserts all **mapped fields** of the current table into the database. We now have to define the status of the new PrimaryKey field for this action.

5. Click OK to confirm the current settings.
6. Right click the **PrimaryKey** item, then select the **Database Key Settings** entry.

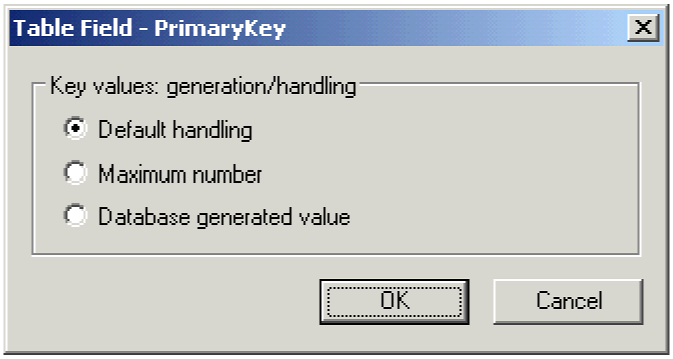


Table Field - PrimaryKey

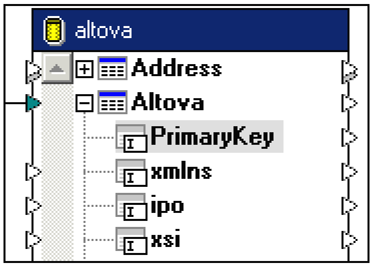
Key values: generation/handling

● Default handling

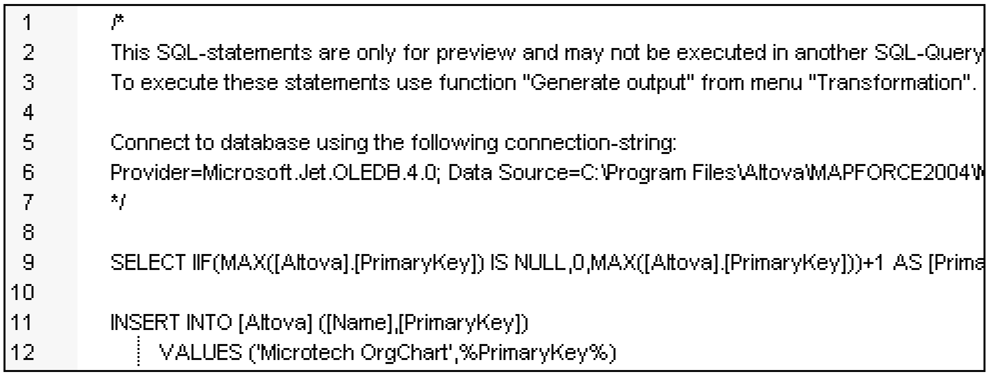
○ Maximum number

○ Database generated value

7. Select the **Maximum number** entry, and click OK to confirm.
   You will notice that the input icon for the PrimaryKey field is now unavailable.

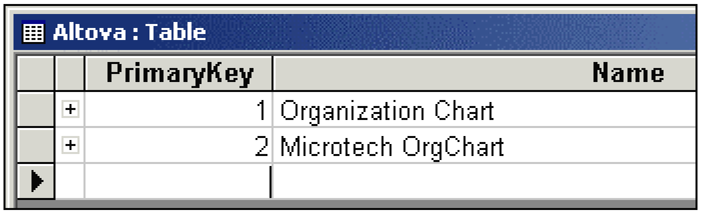


8. Click the Output tab at the bottom of the mapping window to see the pseudo-SQL code that this mapping produces.



```
/*
This SQL-statements are only for preview and may not be executed in another SQL-Query
To execute these statements use function "Generate output" from menu "Transformation".

Connect to database using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MAPFORCE2004\
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [Prima

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
```

9. Click the Run SQL-Script icon ▶ in the function bar to run the script and insert the table data into the database. If the script was successful, a confirmation message appears. Click OK to confirm.
10. Open the Altova database in Access to see the effect.

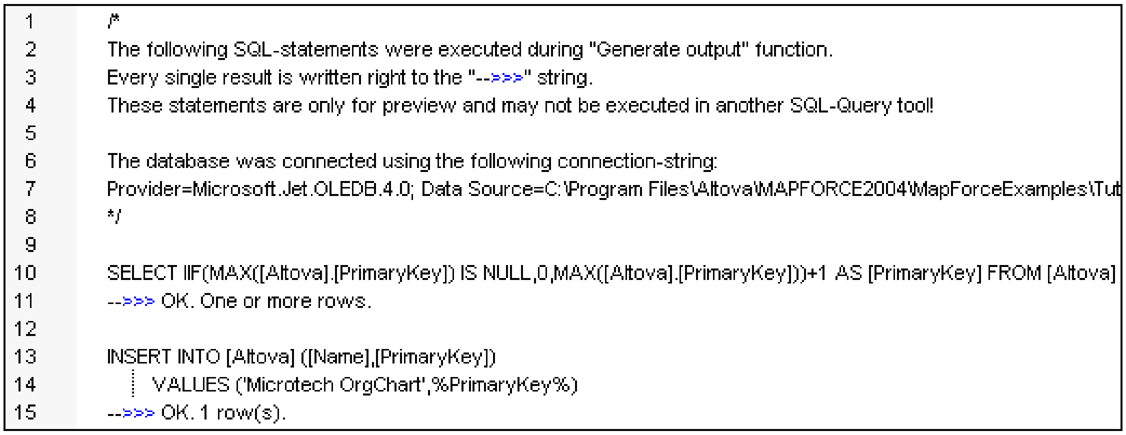


PrimaryKey	Name
1	Organization Chart
2	Microtech OrgChart

A new Microtech OrgChart record has been added to the Altova table with the new PrimaryKey 2. The data for this record originated in the input XML instance.

11. Switch back to MapForce.

    You will now see a record of what happened when the SQL script was processed.



```
/*
The following SQL-statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL-Query tool!

The database was connected using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MAPFORCE2004\MapForceExamples\Tut
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Altova]
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
-->>> OK. 1 row(s).
```

Please note:

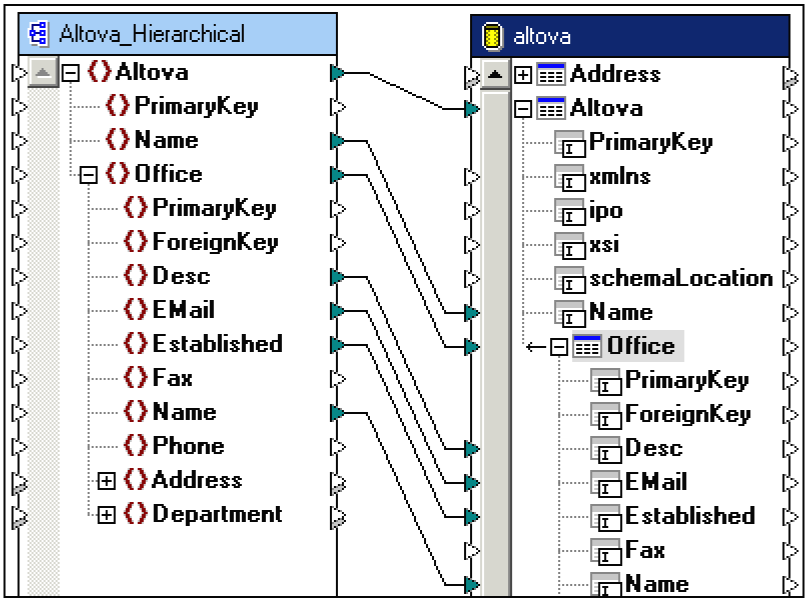
You can only run SQL scripts once from the Output window, you have to switch back to the Mapping window, and to the Output window again, to re-run the script.

Inserting tables and related child tables:

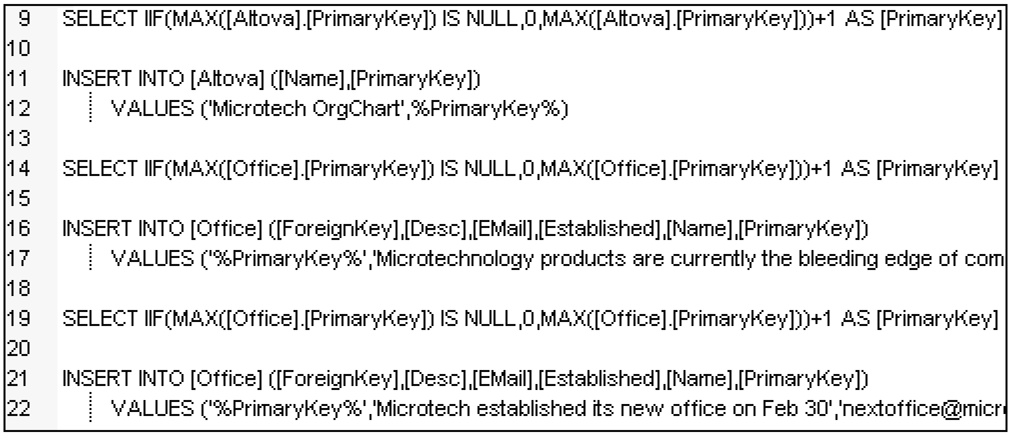
This example uses the previous example as a basis, and extends it by inserting related Office child tables to the Altova parent table.

Table relationships are only generated automatically, when mappings are created **between** child tables of a "root" table. In this case, mappings are created between the Office fields that appear directly under the Altova parent (or "root") table.

1. Right click the **Office** entry and select the menu item "**Database Table Actions**". The Insert All... table action is selected by default, you do not have to make any changes here, click OK to confirm.
2. Right click the **Office | PrimaryKey** field and select the **Database Key Settings** entry.
3. Select the **Maximum number** entry and click OK to confirm.
4. Create the following mappings between the two components:
   Office to Office
   Desc to Desc, and
   Email to Email
   Established to Established, and
   Name to Name.

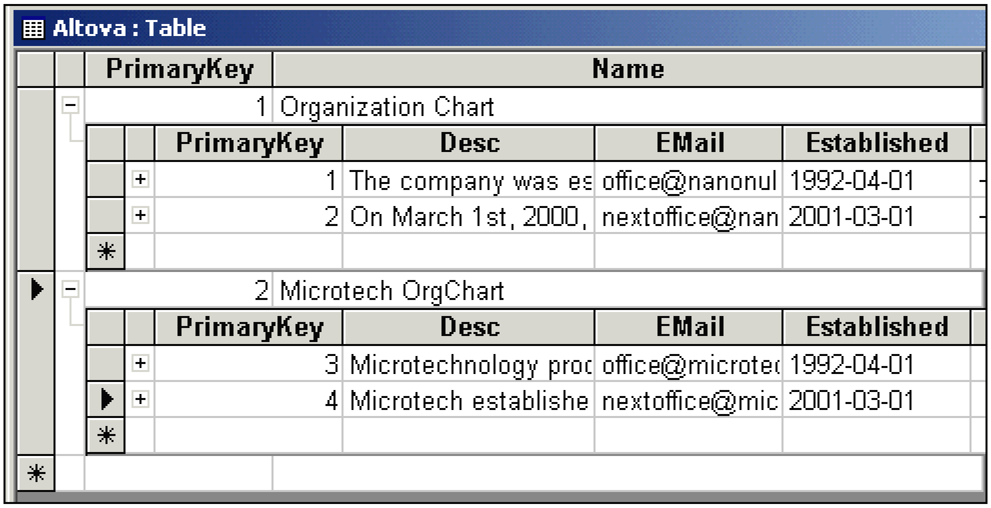


5. Click the Output tab to see the pseudo-SQL code.



```
SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey]

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey]

INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
    VALUES ('%PrimaryKey%','Microtechnology products are currently the bleeding edge of com

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey]

INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
    VALUES ('%PrimaryKey%','Microtech established its new office on Feb 30','nextoffice@micr
```

6. Click the Run SQL script icon to run the script and insert the new tables.
7. Double click the **Altova** table to see the effect in MS-Access.



Altova : Table

PrimaryKey	Name
1	Organization Chart

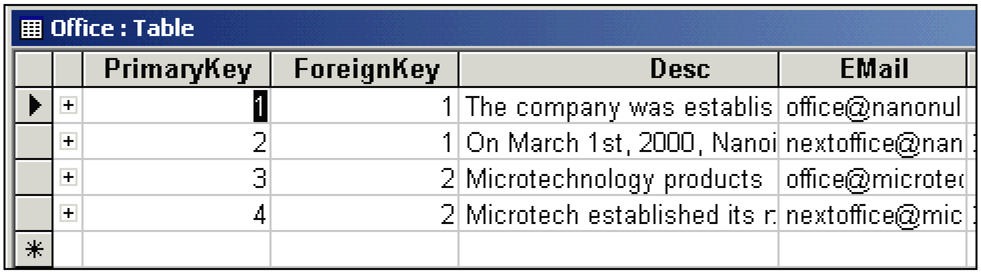
PrimaryKey	Desc	EMail	Established
1	The company was es	office@nanonul	1992-04-01
2	On March 1st, 2000,	nextoffice@nan	2001-03-01

PrimaryKey	Name
2	Microtech OrgChart

PrimaryKey	Desc	EMail	Established
3	Microtechnology proc	office@microtec	1992-04-01
4	Microtech establishe	nextoffice@mic	2001-03-01

Two new offices have been added to the Microtech OrgChart.

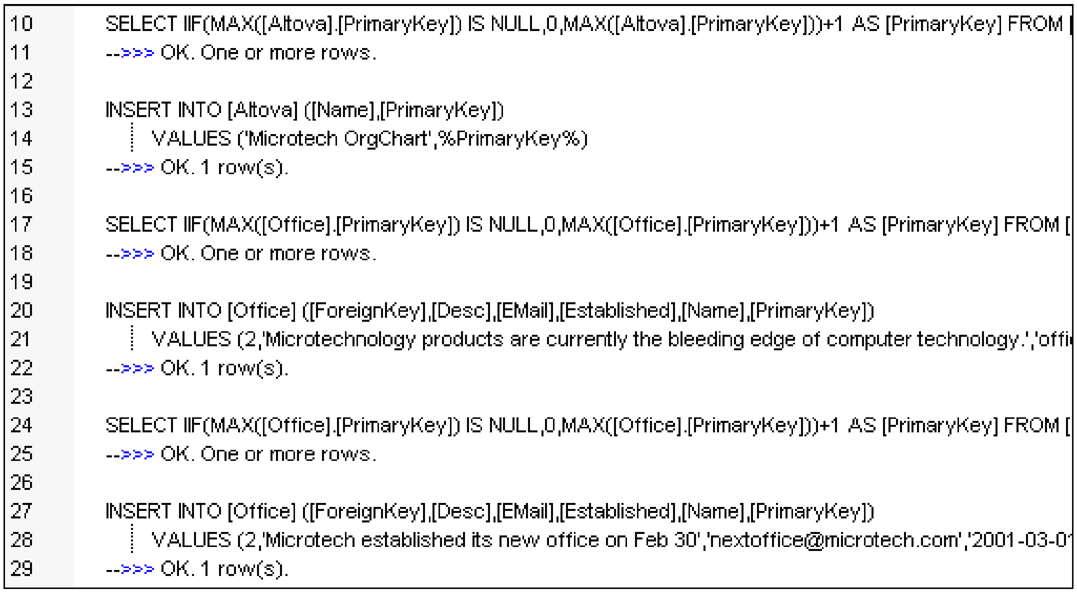
8. Double click the **Office** table to see the effect in greater detail.



Office : Table

PrimaryKey	ForeignKey	Desc	EMail
1	1	The company was establis	office@nanonul
2	1	On March 1st, 2000, Nanoi	nextoffice@nan
3	2	Microtechnology products	office@microtec
4	2	Microtech established its r	nextoffice@mic

The new offices have been added with primary keys of 3 and 4 respectively. Both these new offices are related to the Altova table by their foreign key 2, which references the Microtech OrgChart record.



```
SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM [
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
-->>> OK. 1 row(s).

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM [
-->>> OK. One or more rows.

INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
    VALUES (2,'Microtechnology products are currently the bleeding edge of computer technology.','offi
-->>> OK. 1 row(s).

SELECT IIF(MAX([Office].[PrimaryKey]) IS NULL,0,MAX([Office].[PrimaryKey]))+1 AS [PrimaryKey] FROM [
-->>> OK. One or more rows.

INSERT INTO [Office] ([ForeignKey],[Desc],[EMail],[Established],[Name],[PrimaryKey])
    VALUES (2,'Microtech established its new office on Feb 30','nextoffice@microtech.com','2001-03-0
-->>> OK. 1 row(s).
```

7.3.4 Database action: Update

The first example deals with the simple task of updating existing Person records. Mappings are created from the XML data source to the "root" table Person.

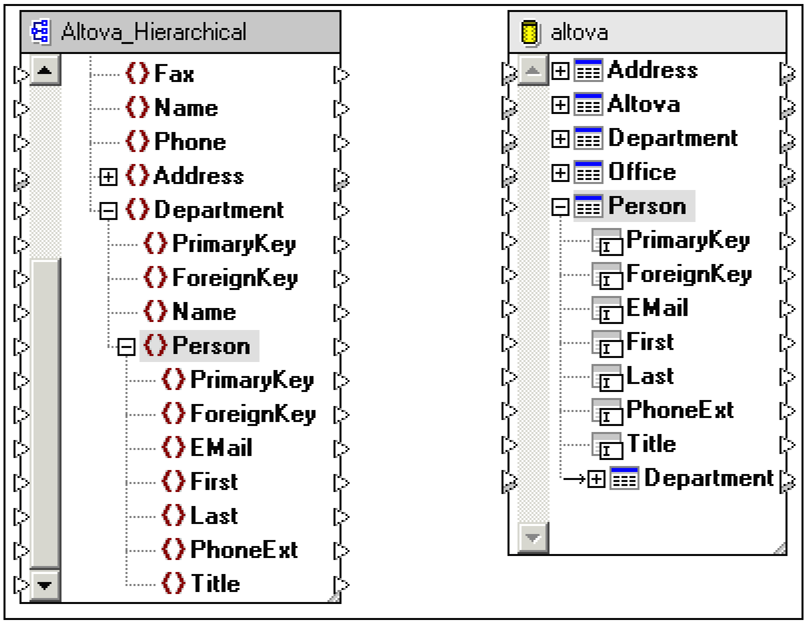
Files used in this example:

- Altova_Hierarchical.xsd
- altova-cmpy.xml
- altova.mdb

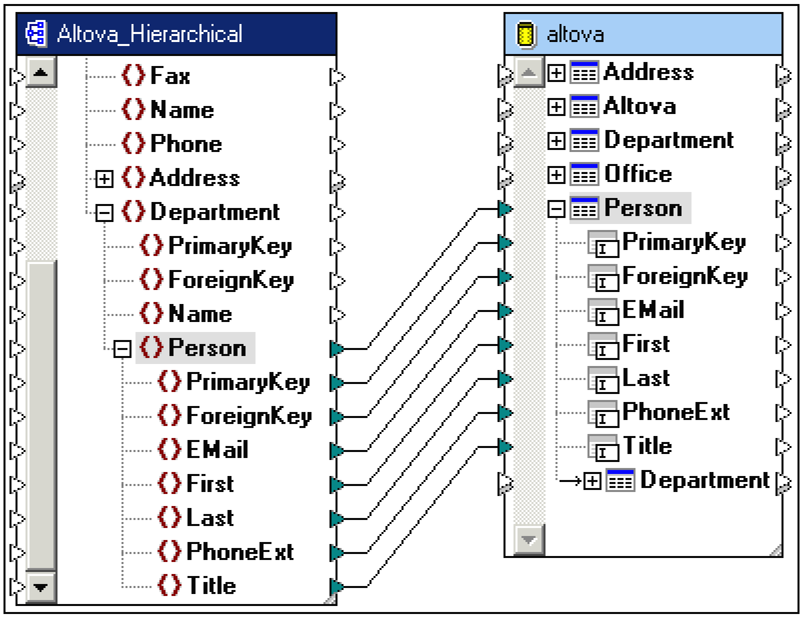
Aim:

To **update** the person fields of the Person table.

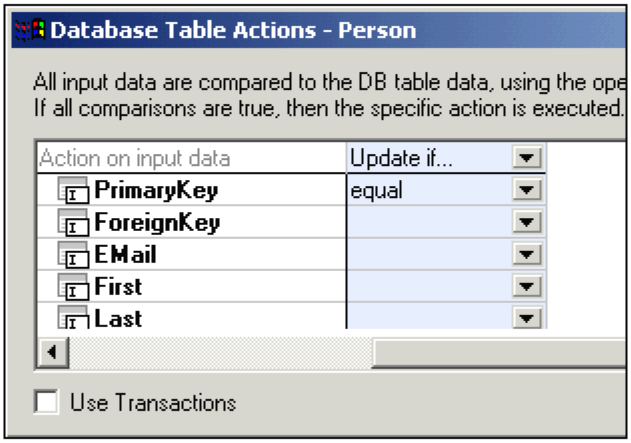
1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.



3. Activate the "Auto connect matching children" icon
4. Click the **Person** item in the XML source file and drag the connector to the Person item of the database. Make sure that you connect to the "**root**" table, Person.
   All matching child items are mapped automatically.



5. Right click the **Person** entry and select the menu item "**Database Table Actions**".
6. Click the Table action combo box, and select **Update if...**
7. Click the combo box in-line with the PrimaryKey entry, and select the **equal** entry, click OK to confirm.



Database Table Actions - Person

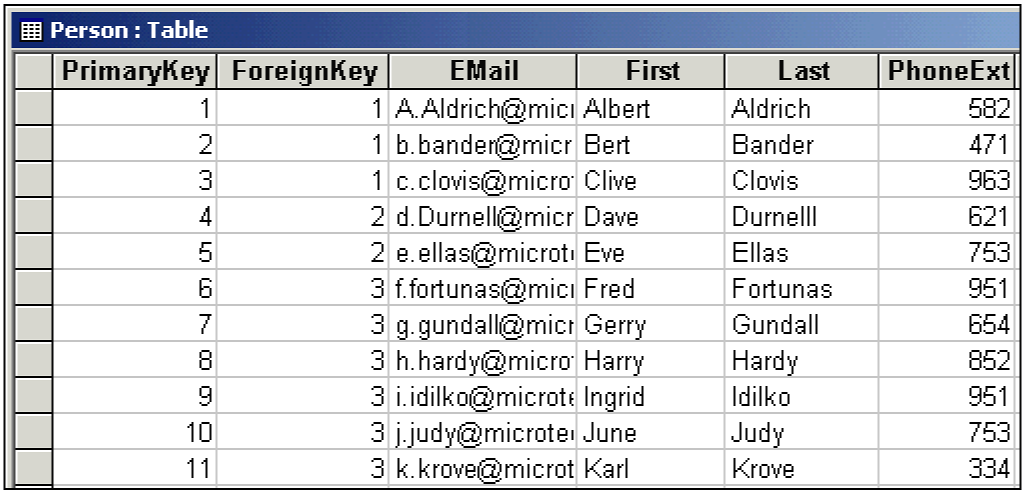
All input data are compared to the DB table data, using the ope
If all comparisons are true, then the specific action is executed.

Action on input data	Update if...
PrimaryKey	equal
ForeignKey	
EMail	
First	
Last	

☐ Use Transactions

The source and target primary keys are compared using the "equal" operator. If the two keys are identical, then all the mapped fields of the Person tables are updated.

8. Click the Output tab at the bottom of the mapping window to see the pseudo-SQL code that this mapping produces.
9. Click the Run SQL-Script icon ▶ in the function bar to run the script and update the database records. If the script was successful, a confirmation message appears. Click OK to confirm.
10. Open the **Altova** database, and double click the **Person** table to see the effect. All the person records of the database have been updated.



Person : Table

PrimaryKey	ForeignKey	EMail	First	Last	PhoneExt
1	1	A.Aldrich@mici	Albert	Aldrich	582
2	1	b.bander@micr	Bert	Bander	471
3	1	c.clovis@micro	Clive	Clovis	963
4	2	d.Durnell@micr	Dave	Durnelll	621
5	2	e.ellas@microt	Eve	Ellas	753
6	3	f.fortunas@mici	Fred	Fortunas	951
7	3	g.gundall@micr	Gerry	Gundall	654
8	3	h.hardy@micro	Harry	Hardy	852
9	3	i.idilko@microte	Ingrid	Idilko	951
10	3	j.judy@microtei	June	Judy	753
11	3	k.krove@microt	Karl	Krove	334

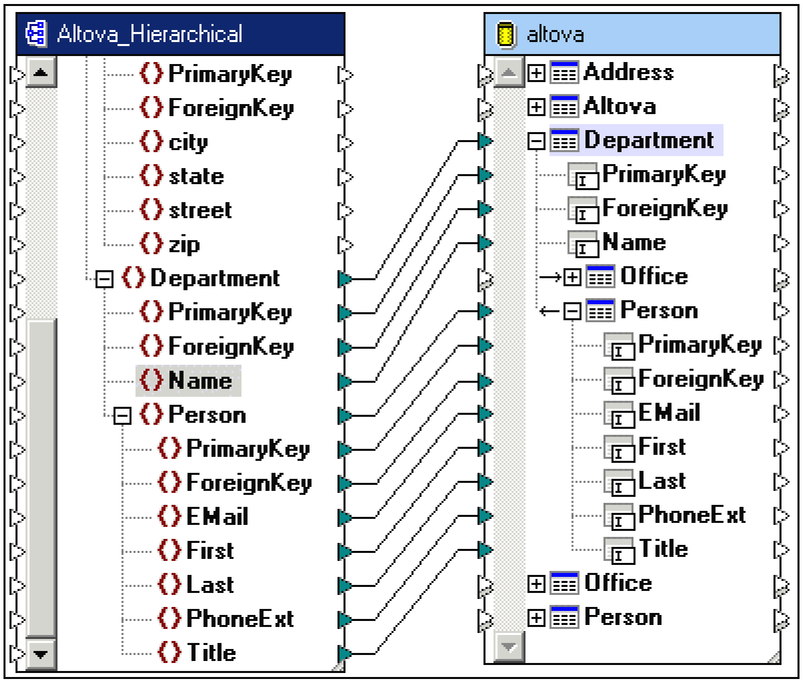
Second Example:
This slightly more complex example, attempts to update records in both the Department and Person tables, as well as add any new Person records which might exist in the XML input file. The "**root**" table used in this example is thus the **Department** table.

Files used in this example:

- Altova_Hierarchical.xsd
- altova-cmpy-**extra**.xml (is the XML instance for Altova_hierarchical.xsd)
- altova.mdb

Aim:

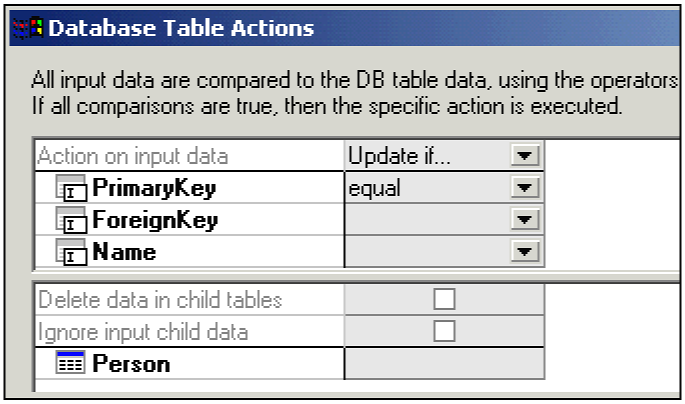
- to **update** the Department Name records
- to **update** existing Person records
- **insert** any new Person records



The source and target primary keys of both tables are compared using the "equal" operator. If the two keys are identical, then the mapped fields of the Department and Person tables are updated. If the comparison fails (in the Person table), then the next table action is processed, i.e. Insert Rest.

Table action: **Department** table

- Table actions **Update if... "equal"** defined for PrimaryKey, i.e. update the Department name if it has changed.



Database Table Actions

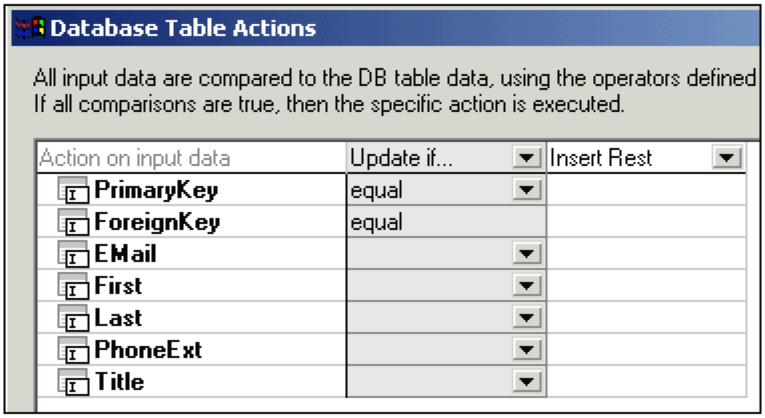
All input data are compared to the DB table data, using the operators
If all comparisons are true, then the specific action is executed.

Action on input data	Update if...
PrimaryKey	equal
ForeignKey	
Name	

Delete data in child tables	□
Ignore input child data	□
Person	

Table action: **Person** table

- Table action **Update if... "equal"** defined for PrimaryKey.

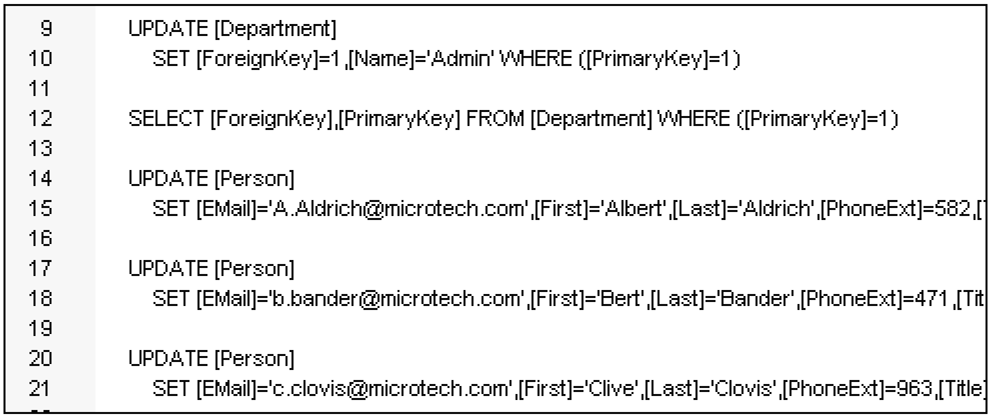


Database Table Actions

All input data are compared to the DB table data, using the operators defined
If all comparisons are true, then the specific action is executed.

Action on input data	Update if...	Insert Rest
PrimaryKey	equal	
ForeignKey	equal	
EMail		
First		
Last		
PhoneExt		
Title		

- Table action **Insert Rest** defined as the second table action should the first comparison, Update if..., fail.
  Click the **Append Action** button to append a new Table action column.



```
UPDATE [Department]
   SET [ForeignKey]=1,[Name]='Admin' WHERE ([PrimaryKey]=1)

SELECT [ForeignKey],[PrimaryKey] FROM [Department] WHERE ([PrimaryKey]=1)

UPDATE [Person]
   SET [EMail]='A.Aldrich@microtech.com',[First]='Albert',[Last]='Aldrich',[PhoneExt]=582,[

UPDATE [Person]
   SET [EMail]='b.bander@microtech.com',[First]='Bert',[Last]='Bander',[PhoneExt]=471,[Tit

UPDATE [Person]
   SET [EMail]='c.clovis@microtech.com',[First]='Clive',[Last]='Clovis',[PhoneExt]=963,[Title]
```

Processing sequence Department table:

Department table: **Update if**... condition **true**:
source and target keys are identical, therefore:

- update each Department record where the keys are identical.
- if records exist in the database with no counterpart in the **source** file, then these records are retained and remain unchanged (in this example the Engineering table).

Department table: **Update if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database,
the update if... condition fails, therefore:

- none of the Department records are updated.

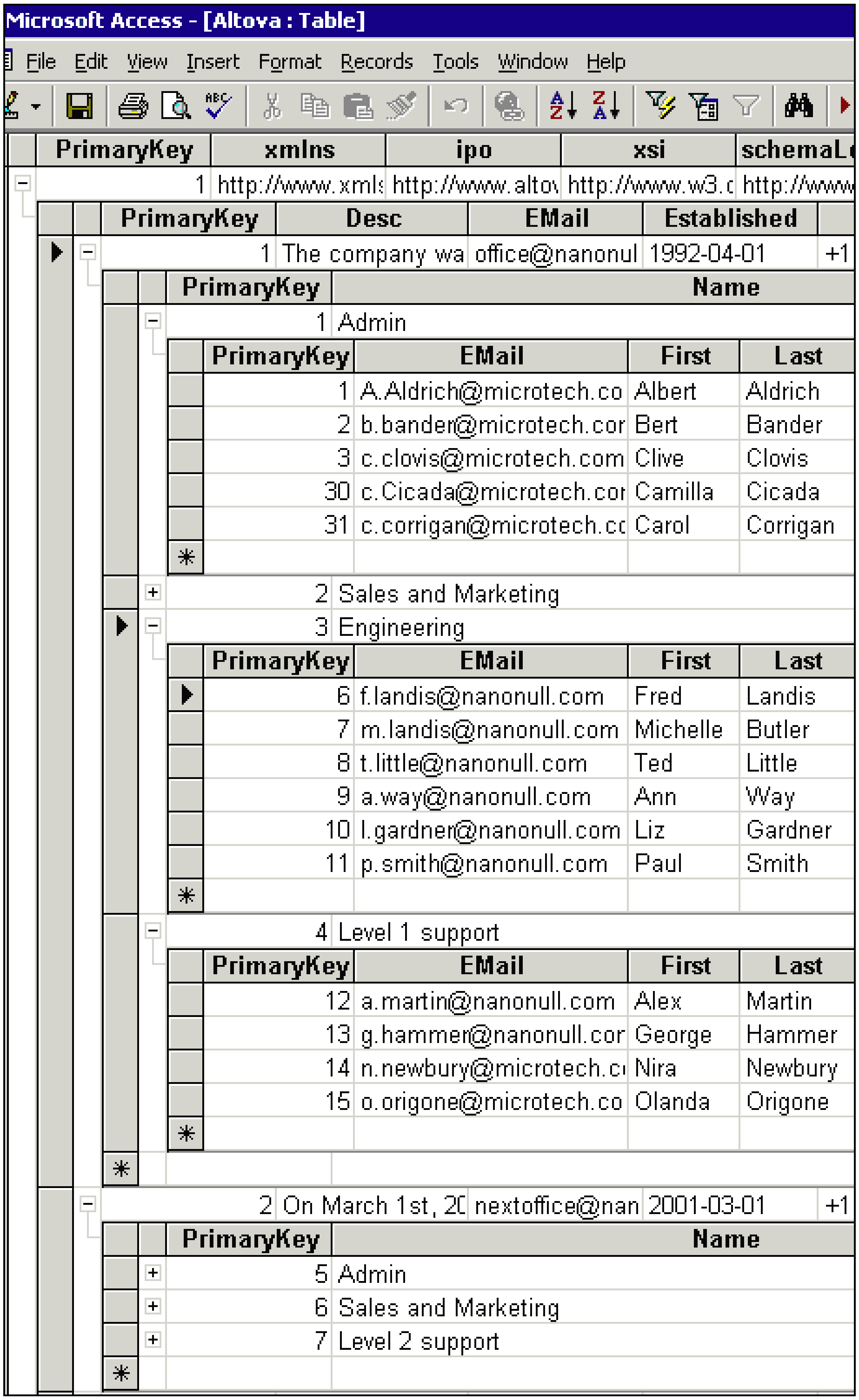
Processing sequence Person table:

Person table: **Update if**... condition **true**:
source and target keys are identical, therefore:

- update each Person record where the keys are identical.
- if records exist in the database with no counterpart in the **source** file, then these records are retained and remain unchanged.

Person table: **Update if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database, the update if... condition fails, therefore:

- move on to the next Table Action column: **Insert Rest**...
- insert the new Person records into the Person table if any exist.
  In this case, two new person records are added to the Admin department, with the person primary keys of 30, and 31, respectively.


Microsoft Access - [Altova : Table]
File Edit View Insert Format Records Tools Window Help

PrimaryKey	xmlns	ipo	xsi	schemaL
1	http://www.xmls	http://www.altov	http://www.w3.c	http://www

PrimaryKey	Desc	EMail	Established	
1	The company wa	office@nanonul	1992-04-01	+1

PrimaryKey	Name
1	Admin

PrimaryKey	EMail	First	Last
1	A.Aldrich@microtech.co	Albert	Aldrich
2	b.bander@microtech.cor	Bert	Bander
3	c.clovis@microtech.com	Clive	Clovis
30	c.Cicada@microtech.cor	Camilla	Cicada
31	c.corrigan@microtech.cc	Carol	Corrigan

PrimaryKey	Name
2	Sales and Marketing
3	Engineering

PrimaryKey	EMail	First	Last
6	f.landis@nanonull.com	Fred	Landis
7	m.landis@nanonull.com	Michelle	Butler
8	t.little@nanonull.com	Ted	Little
9	a.way@nanonull.com	Ann	Way
10	l.gardner@nanonull.com	Liz	Gardner
11	p.smith@nanonull.com	Paul	Smith

PrimaryKey	Name
4	Level 1 support

PrimaryKey	EMail	First	Last
12	a.martin@nanonull.com	Alex	Martin
13	g.hammer@nanonull.cor	George	Hammer
14	n.newbury@microtech.ci	Nira	Newbury
15	o.origone@microtech.co	Olanda	Origone

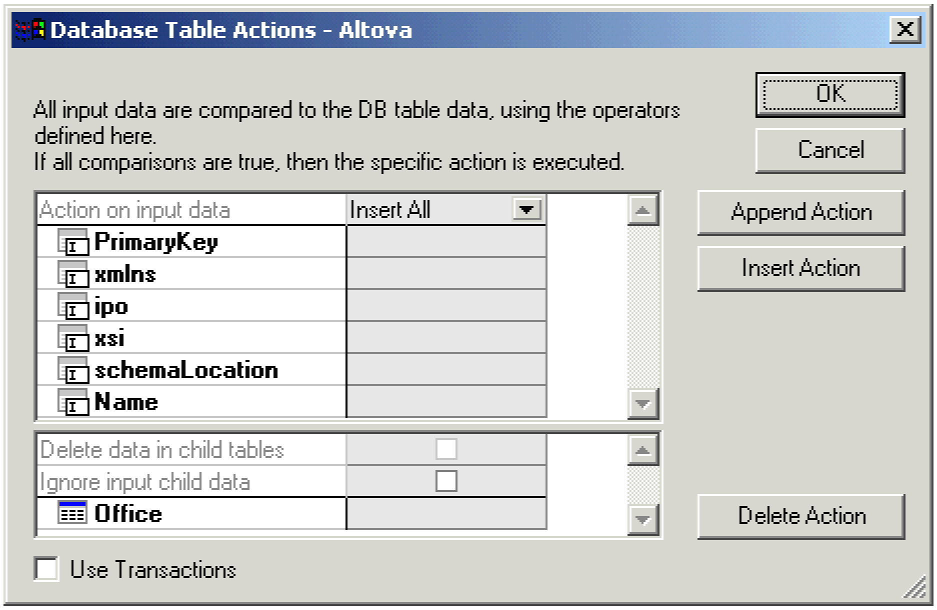
PrimaryKey	Desc	EMail	Established	
2	On March 1st, 2C	nextoffice@nan	2001-03-01	+1

PrimaryKey	Name
5	Admin
6	Sales and Marketing
7	Level 2 support

Update if... combinations - with delete child data

This section describes the effect of the **Update if...** condition on a parent table combined with each of the possible table actions defined for related child tables. The "**Delete data in child**

tables option" is active in all **but one** of these examples. You can continue to use the mapping from the previous section, for this section.



All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on input data	Insert All
PrimaryKey	
xmlns	
ipo	
xsi	
schemaLocation	
Name	

Delete data in child tables	☐
Ignore input child data	☐
Office	

☐ Use Transactions

Files used to illustrate this example:

- Altova_hierarchical.xsd
- Altova-cmpy-**extra**.xml
- Altova.mdb

Update if.. on parent table, **Insert all...** on child table

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	☑		
Ignore input child data	☐		
child table - **Person**			
Table action		**Insert all...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

- Updates parent table data (Department records)
- Deletes child data of those tables which satisfy the Update if... condition (Person records).
  Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).
- Inserts all Person records from the input XML-instance. This also includes new records that might not already exist in the database.

Update if... on parent table, **Update if...** on child table

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey

Delete data in child tables	☑		
Ignore input child data	□		
child table - **Person**			
Table action		**Update if...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

- Updates parent table data (Department records).
- Deletes child data of those tables which satisfy the Update if... condition (Person records).
  Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).
- Update if... condition, defined for the Person table, fails because all Person records in the database have been deleted by the "Delete data in child tables" option. There is no way to compare the database and XML data primary keys, as the database keys have been deleted. No records are updated.

Update if... on parent table, **Delete if...** on child table **(Delete data in child tables - active)**

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	☑		
Ignore input child data	□		
child table - **Person**			
Table action		**Delete if...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

- Updates parent table data (Department records).
- Deletes child data (Person records) from all Departments because the "Delete data in child tables" option is active. All Person records are deleted for each Department which has a corresponding PrimaryKey in the source XML. I.e. even **Person** records of the database which have no counterpart in the source XML, are deleted.
  Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).
- The child table data (Person records) are deleted before the Table action, Delete if..., is executed, no records are deleted.

Update if... on parent table, **Delete if...** on child table **(Delete data in child tables - deactivated)**

Parent table - **Department**			
Table action		Update if...	compare PrimaryKey
Delete data in child tables	□		Delete data... **not active** !
Ignore input child data	□		

child table - **Person**			
Table action		**Delete if...**	compare PrimaryKey
Delete data in child tables			
Ignore input child data			

Result:

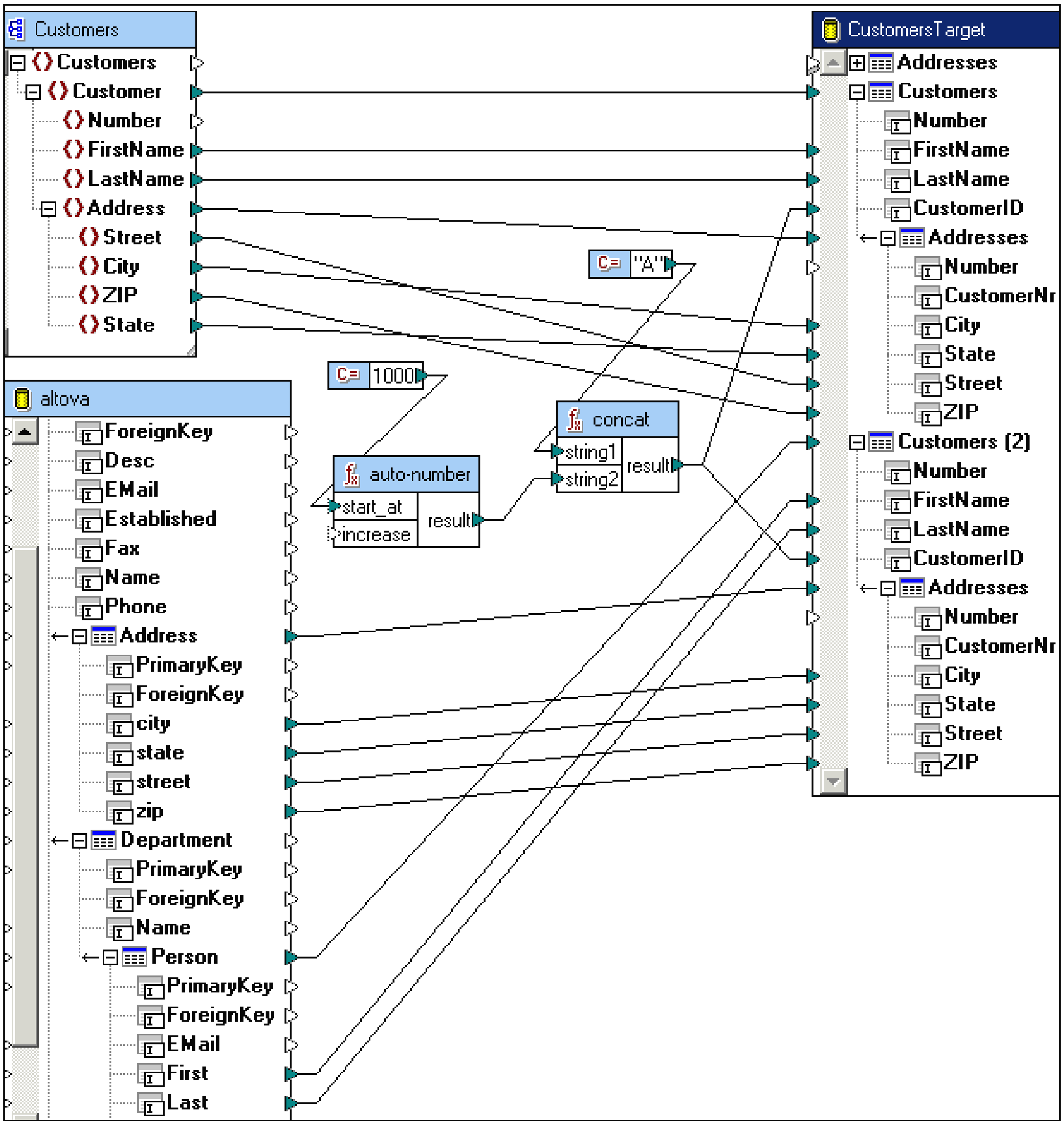
- Updates parent table data (Department records).
  Retains existing database tables (Engineering in this case) which do not have a counterpart in the input XML file (no source keys for the Update if... comparison).
- Delete if... only deletes those Person records for which a corresponding **Person** PrimaryKey exists in the source XML file.
- Database records which do not have the corresponding Person key, are retained.

To see a further example involving duplicate items, Insert, Update and transactions, please see the **Customers_DB.mfd** sample file available in the **...\MapForceExamples** folder. The example shows how XML schemas and database sources can be mapped to target databases.

In the example:

- XML Schema to database:
  Customers and Addresses exist in the target database. These entries are updated with the new data from the from the source XML Schema/document. The FirstName an LastName items are used to find the correct rows in the database.
- Database to database:
  Address and Person data are supplied by the database source and are inserted into the database. The target table (Customers) is duplicated

  CustomerID for each record are created anew, with the initial value being A1000.


Customers
Customers
Customer
Number
FirstName
LastName
Address
Street
City
ZIP
State
altova
ForeignKey
Desc
EMail
Established
Fax
Name
Phone
Address
PrimaryKey
ForeignKey
city
state
street
zip
Department
PrimaryKey
ForeignKey
Name
Person
PrimaryKey
ForeignKey
EMail
First
Last
C= 1000
auto-number
start_at
increase
result
C= "A"
concat
string1
string2
result
CustomersTarget
Addresses
Customers
Number
FirstName
LastName
CustomerID
Addresses
Number
CustomerNr
City
State
Street
ZIP
Customers (2)
Number
FirstName
LastName
CustomerID
Addresses
Number
CustomerNr
City
State
Street
ZIP

7.3.5 Database action: Delete

The table action Delete if... is used to selectively delete data from tables. This is achieved by selecting specific items/fields of the source and target components which are to be compared. The specific table action is then executed depending on the outcome of this comparison.

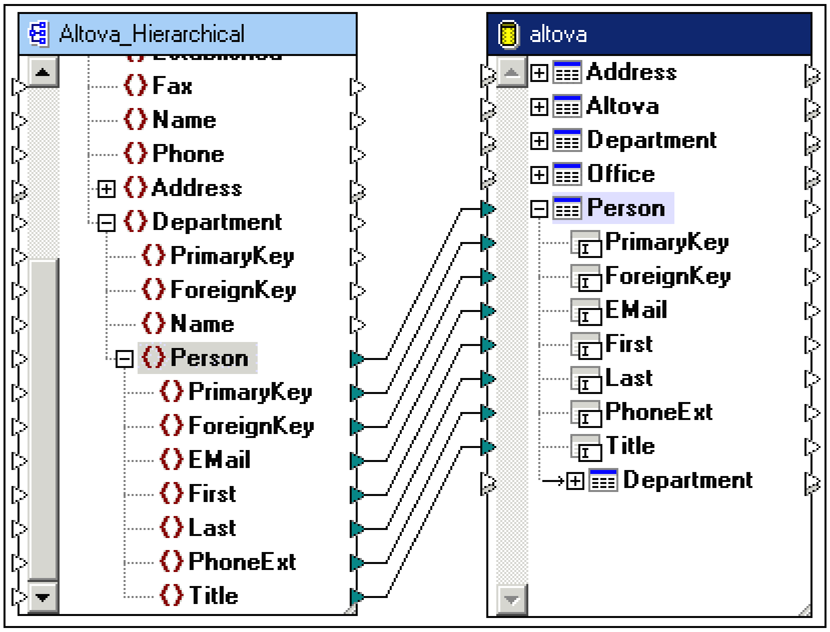
Please note:

This table action should not be confused with the "**Delete data in child tables**" option, available in the table action dialog box. The Delete if... table action only affects the table for which the action is defined, no other tables are affected.

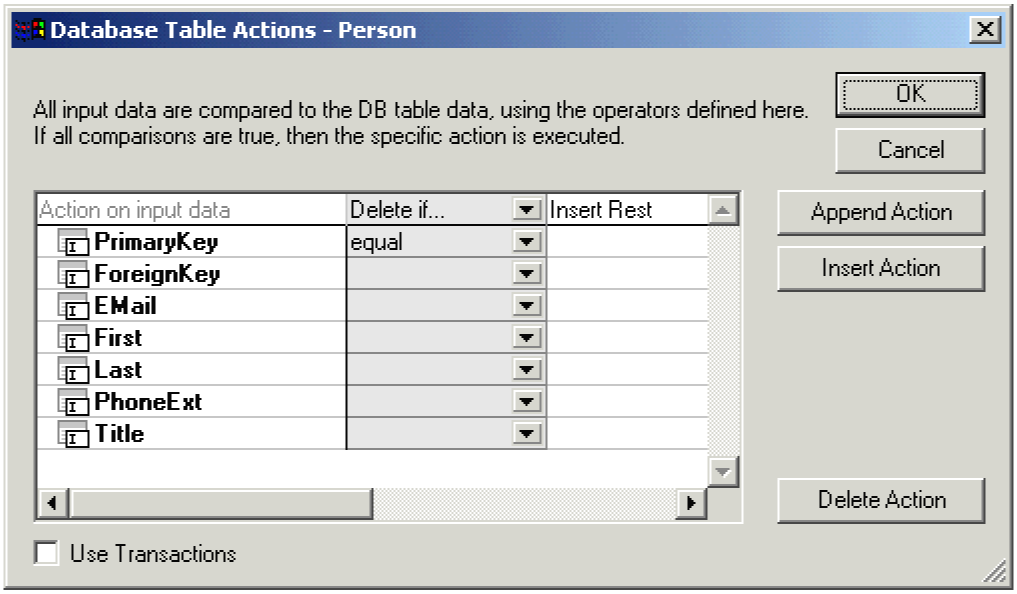
Aim:

- To delete the existing Person records in the database, and
- Insert new Person records from the input XML file.

1. Insert the Altova_Hierarchical schema (and assign **altova-cmpy-extra.xml** as the input XML instance).
2. Insert the MS Access database **altova.mdb** into the mapping.



3. Select the menu option **Connection | Auto Connect matching children**.
4. Click the Person item in the XML source file and drag the connector to the Person item of the database. Make sure that you connect to the "root" table, **Person**.
   All matching child items are mapped automatically.
5. Right click the **Person** entry and select the menu item "**Database Table Actions**".
6. Click the Table action combo box and select **Delete if...**
7. Click the **Append Action** button.
   This automatically inserts a new Table action column with the table action Insert Rest.



Database Table Actions - Person

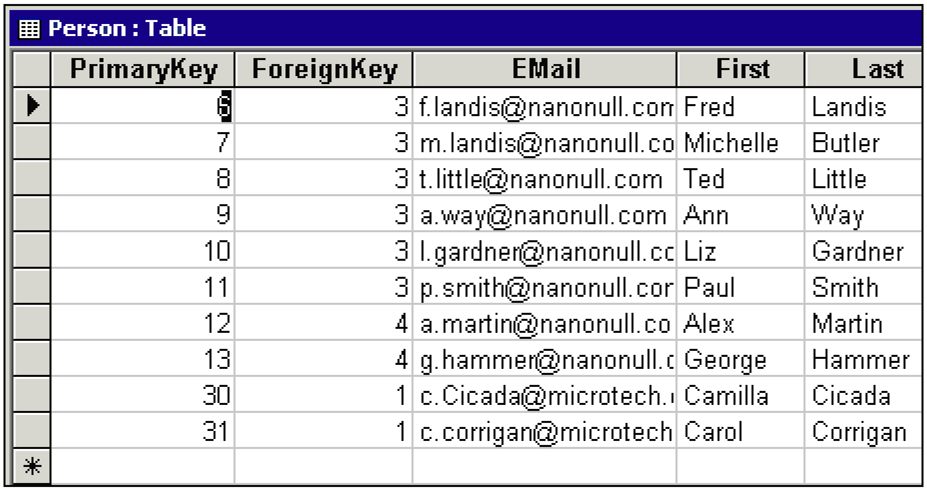
All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on input data	Delete if...	Insert Rest
PrimaryKey	equal	
ForeignKey		
EMail		
First		
Last		
PhoneExt		
Title		

The source and target primary keys are compared using the "equal" operator. If the two keys are identical, then the mapped fields of the Person tables are deleted. Once this has been achieved, the next table action is started, in this case Insert Rest.

Insert Rest inserts all those records, from the source XML file, which do not have a counterpart key/field in the database.

8. Click the Output tab at the bottom of the mapping window to see the pseudo-SQL code that this mapping produces.
9. Click the Run-SQL-Script icon ▶ in the function bar to run the script and update the database records. If the script was successful, a confirmation message appears. Click OK to confirm.
10. Open the Altova database and double click the **Person** table to see the effect.



Person : Table

PrimaryKey	ForeignKey	EMail	First	Last
6	3	f.landis@nanonull.con	Fred	Landis
7	3	m.landis@nanonull.co	Michelle	Butler
8	3	t.little@nanonull.com	Ted	Little
9	3	a.way@nanonull.com	Ann	Way
10	3	l.gardner@nanonull.cc	Liz	Gardner
11	3	p.smith@nanonull.cor	Paul	Smith
12	4	a.martin@nanonull.co	Alex	Martin
13	4	g.hammer@nanonull.c	George	Hammer
30	1	c.Cicada@microtech.	Camilla	Cicada
31	1	c.corrigan@microtech	Carol	Corrigan

Person table: **Delete if**... condition **true**:
source and target keys are identical, therefore:

- delete each Person record where the keys are identical
- if records exist in the database with no counterpart key/field in the source file, then these records are not deleted and remain unchanged.

Person table: **Delete if**... condition **false**:
source and target keys are not identical, i.e. source keys exist which have no match in the target database, the delete if... condition fails, therefore:

- move on to the next Table Action column: **Insert Rest**...
- insert the new Person records into the Person table if any exist.

  In this case, two new person records are added to the Administration department, each with the person primary key of 30, and 31, respectively.

Two additional examples of the Delete if... table action can be viewed in the Update if... combinations section.

7.3.6 Database Key settings

When mapping to databases, MapForce lets you specify how the primary key will be handled. The three options below, are only available if you right click a key field, and select the menu option Database Key settings.

The primary key setting should take the table action defined for that table into account. E.g. when inserting records, the primary key setting should generally be "Maximum number", so that new records are automatically appended to existing ones.

An input icon is only available when "Default handling" is selected. This allows source data to be mapped to the database field directly.

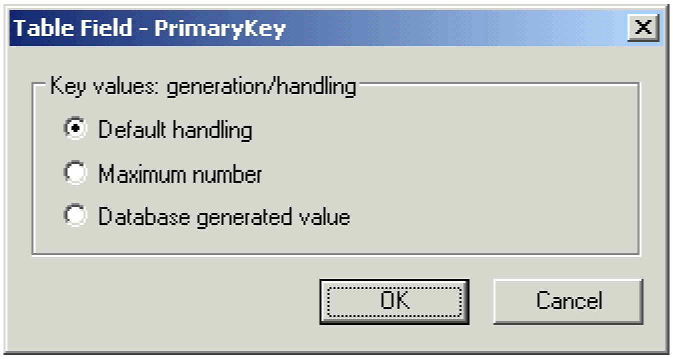


Table Field - PrimaryKey

Key values: generation/handling

● Default handling

○ Maximum number

○ Database generated value

Default handling
This is the standard setting for all database fields.

- an input icon exists when this option is selected, allowing you to map data directly
- the value supplied by the source item, is used as the key value in the database

Maximum number
Use this setting when you want to **Insert** records into the database.

- an input icon is not available, when you select this option.
- the **select** statement **queries** the database for the maximum value of the primary key. This value is then incremented by one and **inserted** into the new field.

Database generated value
Use this setting when the database generates/uses the **Identity function** to generate key values, and you want to **Insert** records.

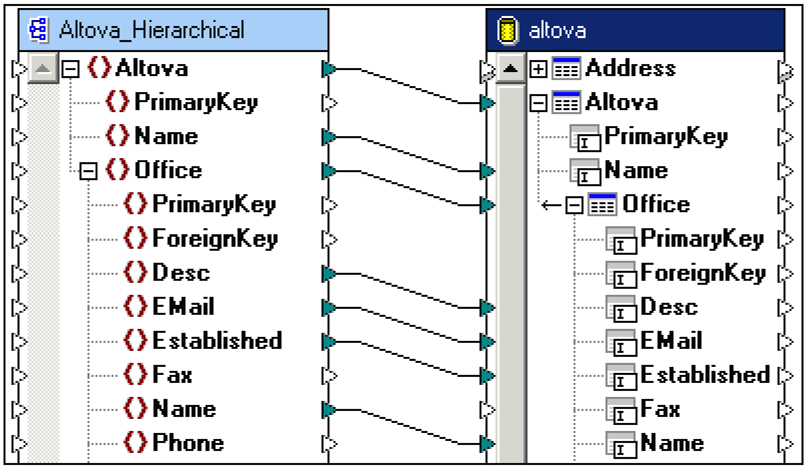
- an input icon is not available, when you select this option.
- the select statement **inserts** the mapped data into the database, **queries** the database for the key value generated by the identity function, and writes it into the key field.

7.3.7 Database Table Actions and transaction processing

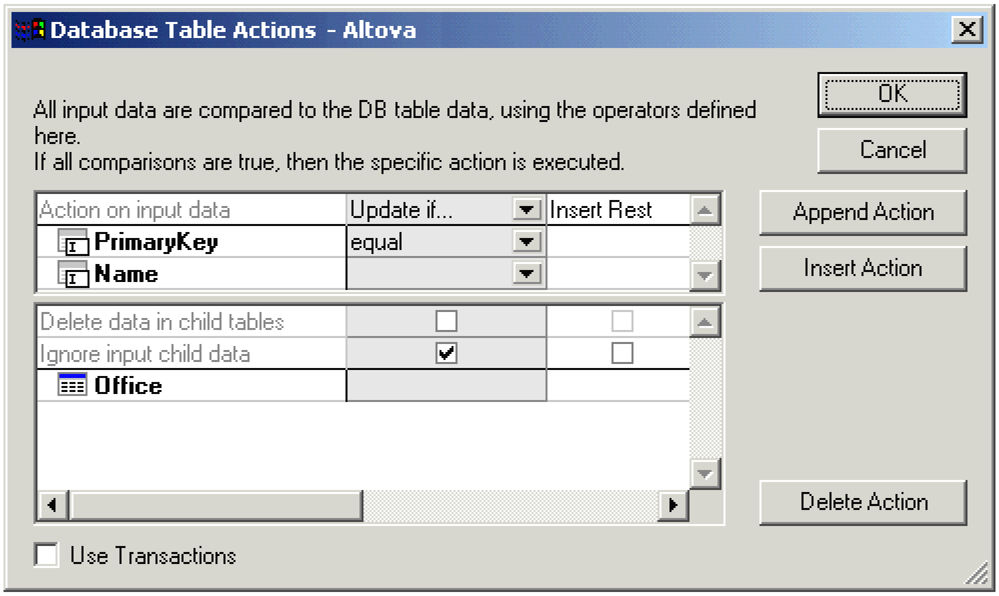
Table actions allow you to define how specific table data are to be manipulated. MapForce currently supports the table actions: insert, update and delete. One or more fields are used to compare source and target data to determine if the table action is to be executed.

The Table Action dialog box allows you to define the:

- fields that will be compared (e.g. PrimaryKey)
- operators used for the comparison (equal, equal ignore case), and
- action taken, when all conditions of each column are fulfilled.



Data may originate from any data source: XML file, database, text, Constant component etc. The mappings that define which data are to be manipulated, are created using connectors in the Mapping window.



All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

Action on input data	Update if...	Insert Rest
PrimaryKey	equal	
Name		

Delete data in child tables	☐	☐
Ignore input child data	☑	☐
Office		

☐ Use Transactions

- Table Actions are processed from left to right. In the example above, the **Update if...** column is processed and then the **Insert Rest...** column.
- **All** the conditions of one column must be satisfied if the table action is to be executed. When this is the case, all those fields are updated where a **mapping exists**, i.e. a

connector exists between the source and target items in the Mapping window.

- If a condition is not satisfied, then the table action for that column is ignored, and the next column is processed.
- If none of the conditions are "true", no table action takes place.

Delete data in child tables:

- Standard setting when you select the **Update if...** action.
- Necessary if the no. of records in the source file might be different from the no. of records in the target database.
- Helps keep the database synchronized (no orphaned data in child tables)

Effect:

- The Update if... condition is satisfied when a corresponding key (or any other field) exists in the source XML file. All **child data** of the parent table are deleted.
- Update if... selects the parent table, and thus the child tables related to it, on which the "Delete data in child tables" works.
- If the update condition (on the parent) is not satisfied, i.e. no corresponding key/field in source XML file exists, then child data are not deleted.
- Existing database records, that do not have a counterpart in the source file, are not deleted from the database, they are retained.

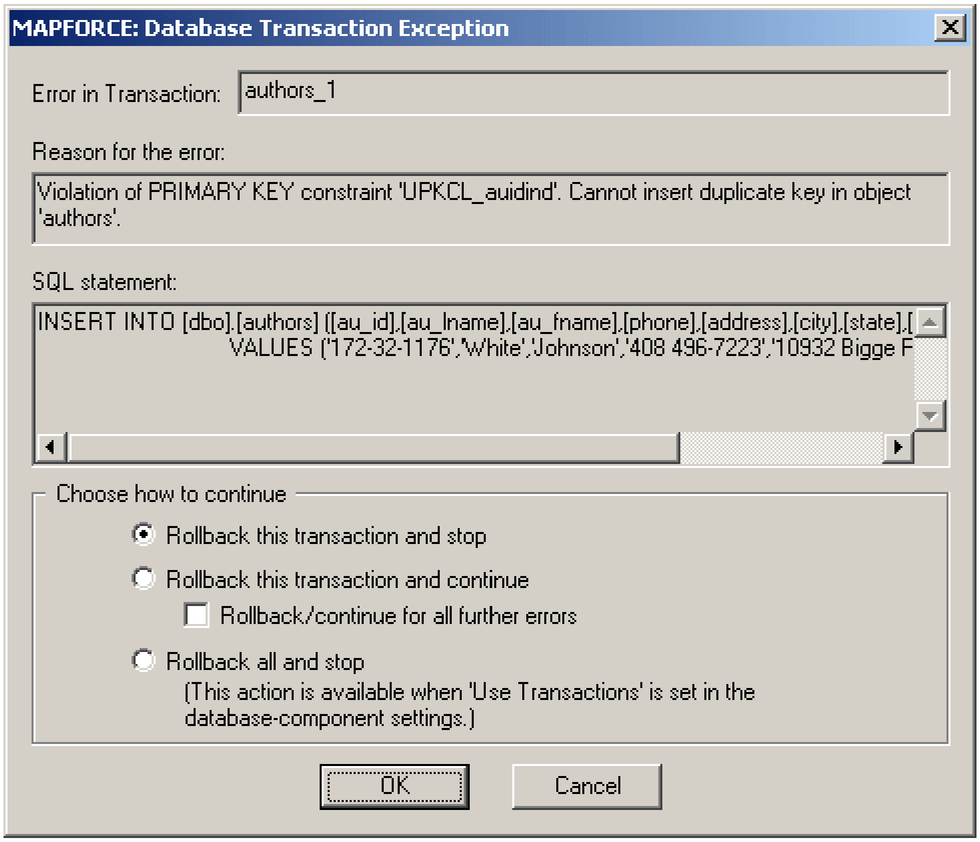
Ignore input child data:
Use this option when you want to update specific table data, without affecting any of the child tables/records of that table.

For example, your mapping setup might consist of 3 source records and 2 target database records.
You would therefore need to:

- define an **Update if...** condition, to update the existing records
- activate the **Ignore input child data** check box, of the **Update if... column**, to ignore the related child records, and
- define an **Insert Rest...** condition for any new records, that have to be inserted.

Use Transactions:
The "Use Transaction" check box allows you to define what is to happen if a database action does not succeed for whatever reason. When such an exception occurs, a dialog box opens prompting you for more information on how to proceed. You then select the specific option and click OK to proceed. Activating this option for a specific table (using the table action dialog box), allows that specific database table to be rolled back when an error occurs.

The transaction setting can also activated for the database component, by right clicking it, in the Component Settings dialog box of the respective database component. In this case, all tables can be rolled back.



Error in Transaction: authors_1

Reason for the error:
Violation of PRIMARY KEY constraint 'UPKCL_auidind'. Cannot insert duplicate key in object 'authors'.

SQL statement:

```
INSERT INTO [dbo].[authors] ([au_id],[au_lname],[au_fname],[phone],[address],[city],[state],[
                      VALUES ('172-32-1176','White','Johnson','408 496-7223','10932 Bigge F
```

Choose how to continue

● Rollback this transaction and stop

○ Rollback this transaction and continue

□ Rollback/continue for all further errors

○ Rollback all and stop
(This action is available when 'Use Transactions' is set in the database-component settings.)

No Transaction options set:
If the transaction check box has not been activated in the table options, or in the component settings, and an error occurs:

- Execution stops at the point the error occurs. All previously successful SQL statements are executed and the on-trues are stored in the database.

Transaction option set at **database component** level:

- Execution stops at the point the error occurs. All previously successful SQL statements are rolled back. No changes are made in the database. All previously successful SQL statements for that for the **database** and all its tables can be rolled back.

Transaction option set at **Table Actions** level:

- The Transaction exception dialog box appears with the "Rollback all and stop" option **disabled**. The failed SQL statement for that specific **table** can be rolled back.

Transaction option set at both **database component** and **table action** level:

- The Transaction exception dialog box appears with the "Rollback all and stop" option **enabled**. All previously successful SQL statements for that for the **database** and all its tables can be rolled back.

Hitting the **Cancel** button, rolls back the current SQL statement and stops.

Please note:
The transaction prompts are only displayed when the transformation is performed **interactively!**

Generated code performs a rollback (and stop) when the first error is encountered.

7.3.8 Generating output values

The Java, C++, and C# libraries have been extended by two functions which can generate values for database fields, which do not have any input data from the Schema, or database.database.

Auto-number and create-guid can both generate values for fields. Both functions are located in the **generator functions** subset of the **lang** library.

auto-number
is generally used to generate primary key values for a numeric field.

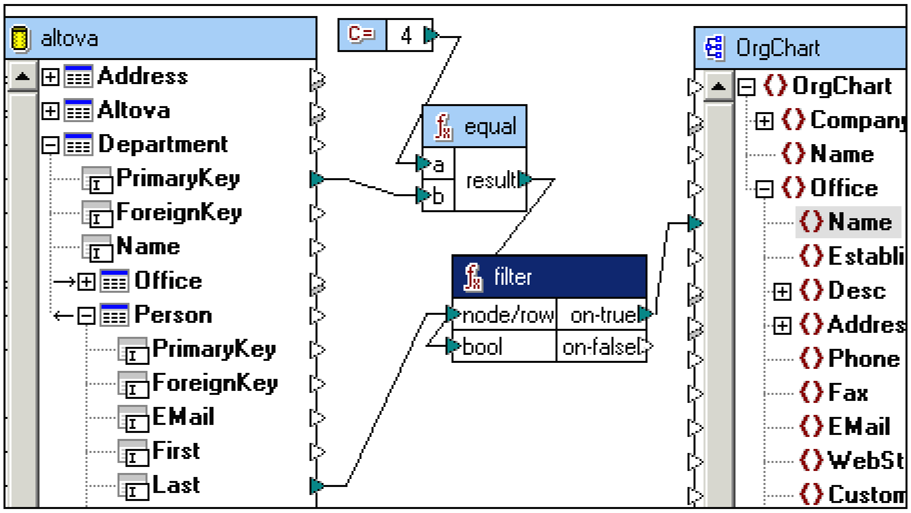
create-guid
Creates a globally-unique identifier (as a hex-encoded string) for the specific field.

7.4 Database relationships and how to preserve or discard them

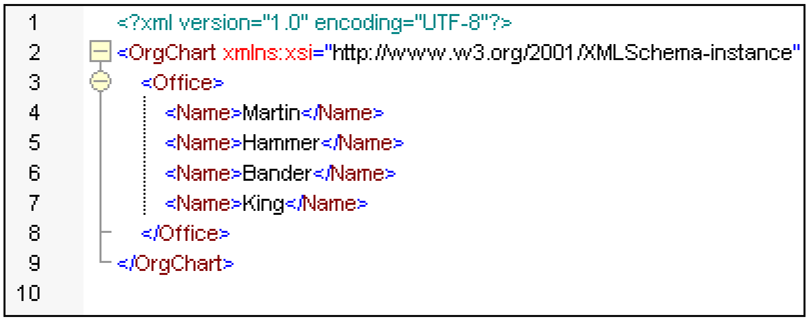
Maintaining database relationships

To map all Last names from the Person table, where the Department primary key is equal to 4, create mappings under the **same** "root" table, Department:

- **PrimaryKey** is mapped from the Department "root" table.
- **Last** is mapped from the Person table, **which is a child** of the Department "root" table.



Result of the above mapping:
The last names of the persons in the Department with the primary key of 4, (IT & Technical support), are displayed in the Output tab.

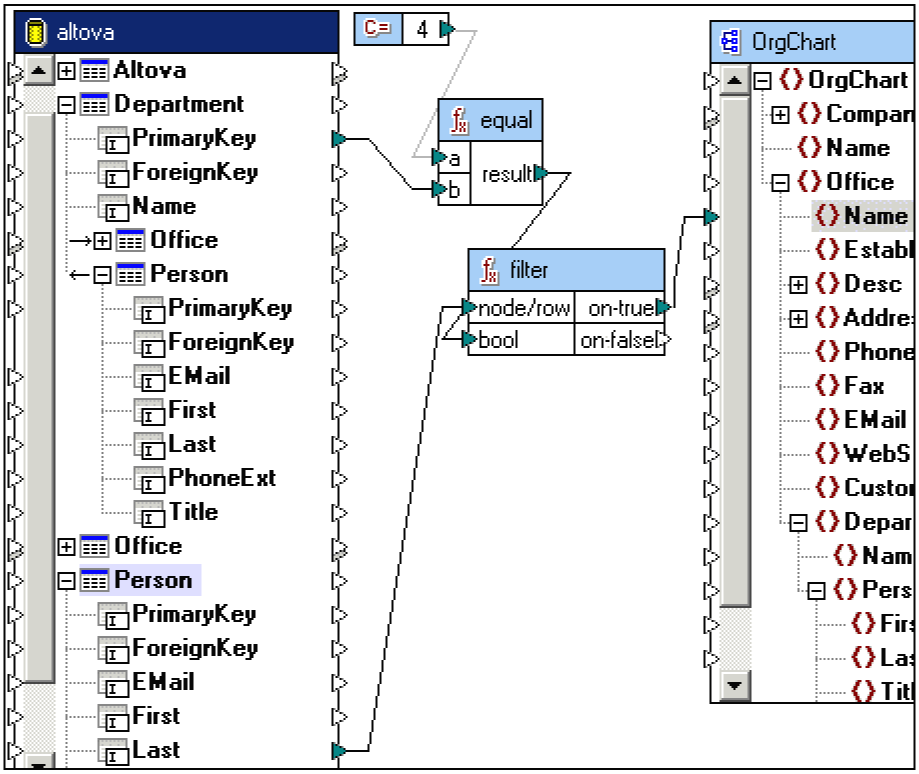


```
<?xml version="1.0" encoding="UTF-8"?>
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Office>
    <Name>Martin</Name>
    <Name>Hammer</Name>
    <Name>Bander</Name>
    <Name>King</Name>
  </Office>
</OrgChart>
```

Discarding database relationships

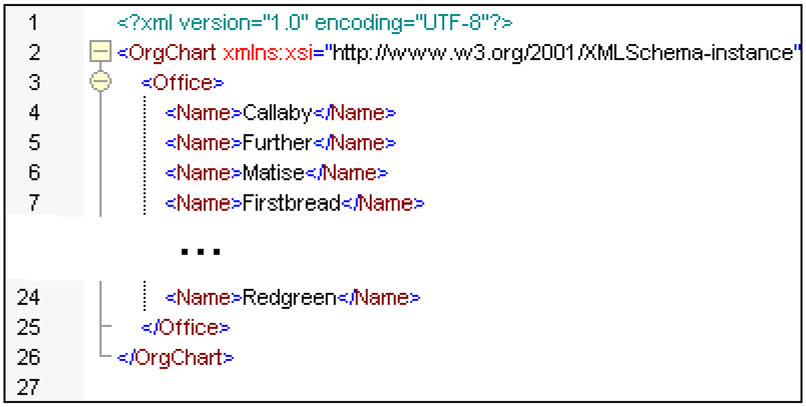
Create mappings of the same fields from **different** root tables (e.g. Department and Office)

- **PrimaryKey** is mapped from the Department "root" table.
- **Last** is mapped from the Person "root" table.



Result of the above mapping:
This mapping method does not deliver the same result, as the table dependencies between the Department and Person tables are now not taken into account.

The result contains the last names of all 21 persons in the database, the **filtering** by the Department primary key has clearly not succeeded.



```
<?xml version="1.0" encoding="UTF-8"?>
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Office>
        <Name>Callaby</Name>
        <Name>Further</Name>
        <Name>Matise</Name>
        <Name>Firstbread</Name>

        ...

        <Name>Redgreen</Name>
    </Office>
</OrgChart>
```

7.5 Database feature matrix

The following tables supply information on the mapping capabilities of MapForce vis-a-vis the major database types.

The following information is supplied:

- General info relating to Database as service, and authentication issues
- Supported connection types
- SQL support for: schemas, join statements etc.
- Transaction methods supported

7.5.1 Database info - MS Access

	MS Access	supported	Notes
General:			
	DB engine as service	n	implemented in OLEDB-provider or ODBC-driver
	own authentication	y	authentication is possible
	Trusted authentication	n	
Connection:			
	OLE DB	y	
	OLE DB connection-string issues	none	
	ODBC	y	
	ODBC connection-string issues	DBQ	must be applied
	ODBC connection-string issues	DATABASE	must not be applied
	JDBC	y	via ODBC
	JDBC URL issues	none	
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	[] or ""	
	support for DB-schemas	n	not supported by Access
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	y	
	JOIN support	n	limited support
	MF: upper function	Ucase(..)	
SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	n	
	retrieve parameter types	?	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	not supported by Access
	set transaction isolation	n	not supported by Access
	MF: begin transaction	API-call	

	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	--	
	MF: rollback to save point	--	
	MF used init. Statements	none	

7.5.2 Database info - MS SQL Server

	MS SQLServer	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	y	
Connection:			
	OLE DB	y	
	ODBC	y	
	ODBC connection-string issues	Select Method=Cursor	must be applied
	JDBC	y	
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	[] or ""	
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution:			
	exec. multiple-stat. in one	y	
	command separator	';' or 'GO'	
	special error handling	n	
	retrieve parameter types	y	with limits
	special issues when using ?	y	DATETIME datatype not supported when using ODBC
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	y	a MUST when using nested transactions. Mixing API-transaction handling and SQL-transaction-commands is not possible
	Nested transactions supported	y	via SAVEPOINTS
	set transaction isolation	y	
	MF: begin transaction	BEGIN TRANSACTION	
	MF: commit transaction	COMMIT TRANSACTION	
	MF: rollback transaction	ROLLBACK TRANSACTION	

	MF: set save point	SAVEPOINT	
	MF: rollback to save point	ROLLBACK TO	
	MF used init. Statements	none	

7.5.3 Database info - Oracle

	Oracle	**supported**	**Notes**
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	n	
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	ODBC connection-string issues	DATABASE	must not be applied
	JDBC	y	
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	""	
	support for DB-schemas	y	
	identity support	n	must use triggers
	MF: read back identity value	not supported	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution :			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	n	
	retrieve parameter types		
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	y	via SAVEPOINTS
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	SAVEPOINT	
	MF: rollback to save point	ROLLBACK TO SAVEPOINT	

	MF used init. Statements	none	

7.5.4 Database info - MySQL

	MySQL	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	n	
	special issues	TYPE=INNODB	for tables when relations, transactions, are used
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	JDBC	y	via ODBC
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	``	
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	n	special implementation for DELETE necessary
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	n	
	retrieve parameter types	y	with limits
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	MySQL does not produce an error, and continues if no nested transactions exist
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	SAVEPOINT	

	MF: rollback to save point	ROLLBACK TO	
	MF used init. Statements	SET AUTOCOMMIT=0	

7.5.5 Database info - Sybase

	Sybase	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	
	Trusted authentication	n	
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	ODBC connection-string issues	Select Method=Cursor	must be applied
	JDBC	y	via ODBC
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification		
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	@@IDENTITY	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution:			
	exec. multiple-stat. in one	y	
	command separator	none	
	special error handling	n	
	retrieve parameter types		
	special issues when using ?	y	only ASCII-127 characters are allowed in string constants when using ODBC
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	not supported by MAPFORCE	
	Nested transactions supported	n	Sybase does not produce an error, continues if no nested transactions exist
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	

	MF: rollback transaction	API-call	
	MF: set save point	SAVE TRANSACTION	
	MF: rollback to save point	ROLLBACK	
	MF used init. Statements	none	

Having defined relationships between tables using the Sybase 'sp_primarykey' and 'sp_foreignkey' procedures, it is additionally necessary to use ALTER TABLE to add a constraint to the table describing the foreign key relationship to have the primary/foreign relationships appear in MapForce.

7.5.6 Database info - IBM DB2

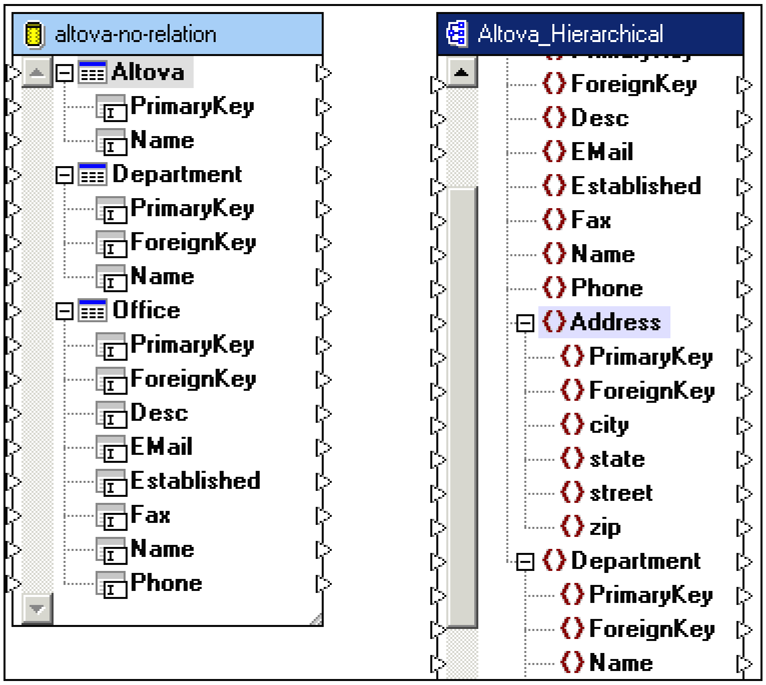
	IBM DB2	supported	Notes
General:			
	DB engine as service	y	
	own authentication	y	uses local windows user-accounts
	Trusted authentication	y	see 'own authentication'
Connection:			
	OLE DB	n	not supported by MapForce
	ODBC	y	
	JDBC	y	via ODBC
	MF used init. Statements	none	
	MF used final. Statements	none	
SQL:			
	DB-object-name qualification	""	
	support for DB-schemas	y	
	identity support	y	
	MF: read back identity value	identity_val_local()	
	sub select support	y	
	JOIN support	y	
	MF: upper function	UPPER()	
SQL-Execution:			
	exec. multiple-stat. in one	n	
	command separator	--	
	special error handling	y	
	retrieve parameter types	n	
	special issues when using ?	n	
Transactions:			
	Flat transactions supported	y	
	Start flat-transaction via execution of SQL-command	n	
	Nested transactions supported	n	not supported by DB2
	set transaction isolation	y	
	MF: begin transaction	API-call	
	MF: commit transaction	API-call	
	MF: rollback transaction	API-call	
	MF: set save point	--	not supported by DB2
	MF: rollback to save point	--	not supported by DB2
	MF used init. Statements	none	

7.6 Using MapForce to create database relationships

MapForce allows you to extract related database data, even if no such relationships exist in the source database.

- any database fields can be used as primary or foreign keys
- new relations can be created that do not currently exist in the database

The MS Access **altova-no-relation.mdb** database used in this example, is a simplified version of the Altova.mdb database supplied with MapForce. The Person and Address tables, as well as all table relationships have been removed in MS Access.

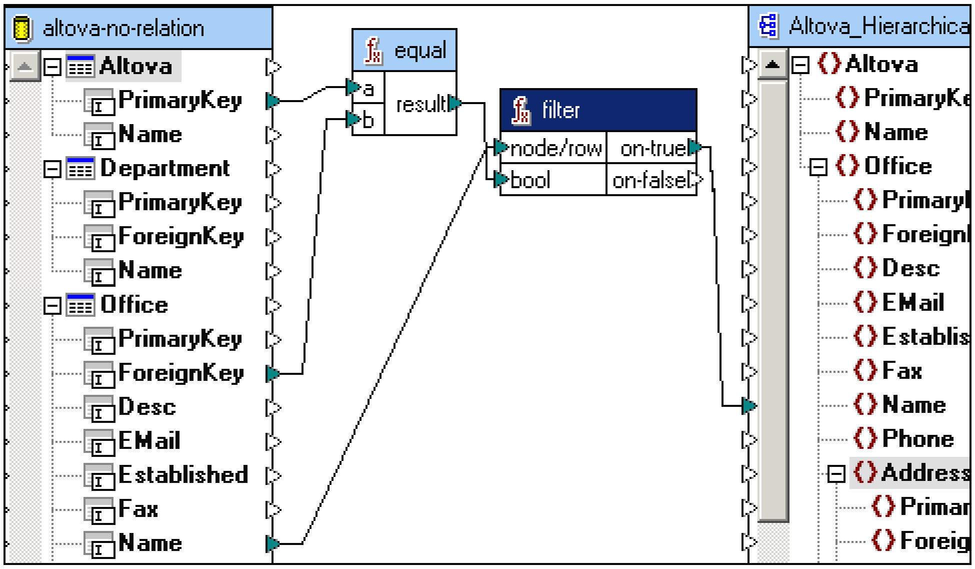


None of the tables visible in the altova-no-relation tree have any child tables, all tables are on the same "**root**" level. The content of each table is limited to the fields it contains. We can however, use MapForce to extract related database data, even though relationships have not been explicitly defined.

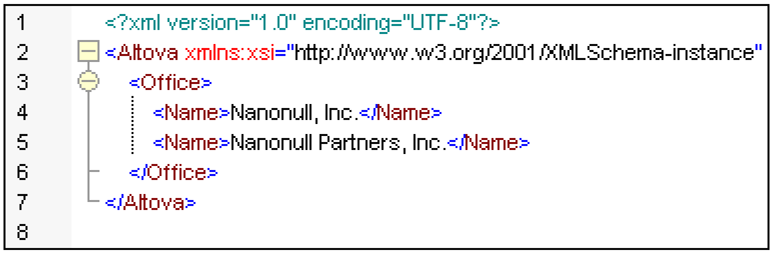
Having defined mappings, switching to the Output tab, allows you to preview the result of your mappings immediately. Database data cannot be previewed if the target language is XSLT, a message will appear and the database component will be greyed out.

In this example we want to retrieve the Altova office names, as well as the departments that exist in each office.

- Compare the **Office foreign** key to the **Altova primary** key, using the "equal" component. If both keys are the same, then
- Pass on the contents of **Office**/Name to the **node/row** parameter of the filter component, and
- Place the on-true result in the Office/Name item in the target schema/document.



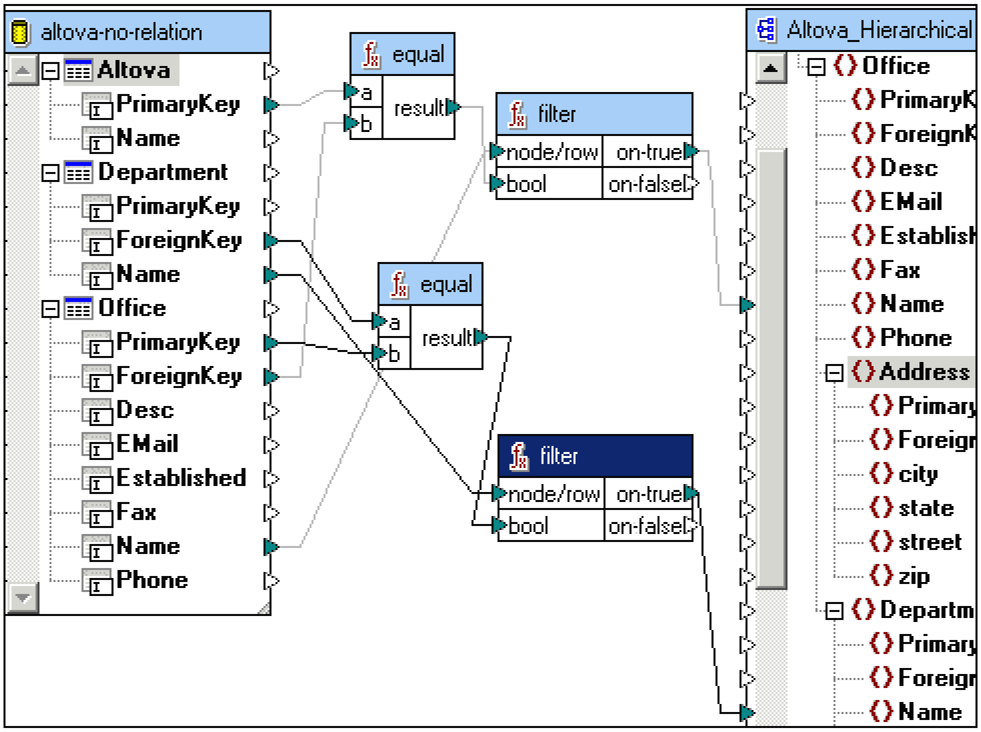
The result of the above mapping is shown below, both Altova office names appear under the Office element.



```
<?xml version="1.0" encoding="UTF-8"?>
<Altova xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Name>Nanonull Partners, Inc.</Name>
  </Office>
</Altova>
```

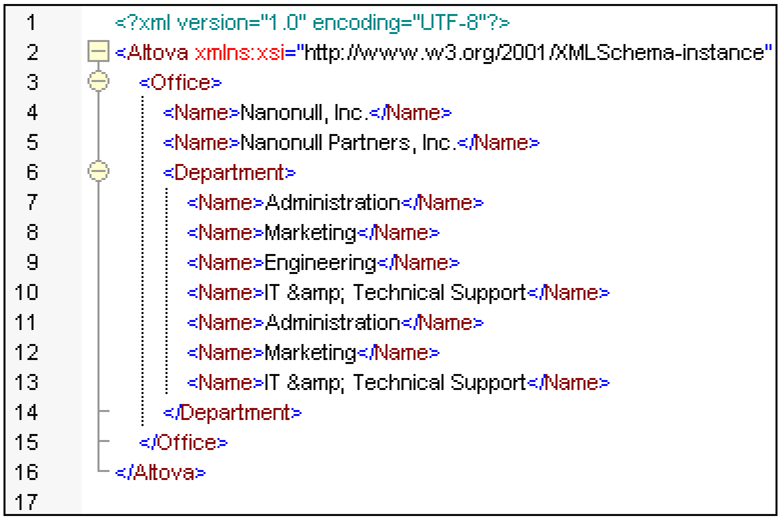
The next bit of information we want to extract are the department names of each office:

- Compare the **Department foreign** key with the **Office primary** key. If both keys are the same, then
- Pass on the contents of the **Department**/name to the **node/row** parameter of the filter component, and
- Place the on-true result in the Department/Name item in the target schema/document.



The result of the above mapping is shown below:

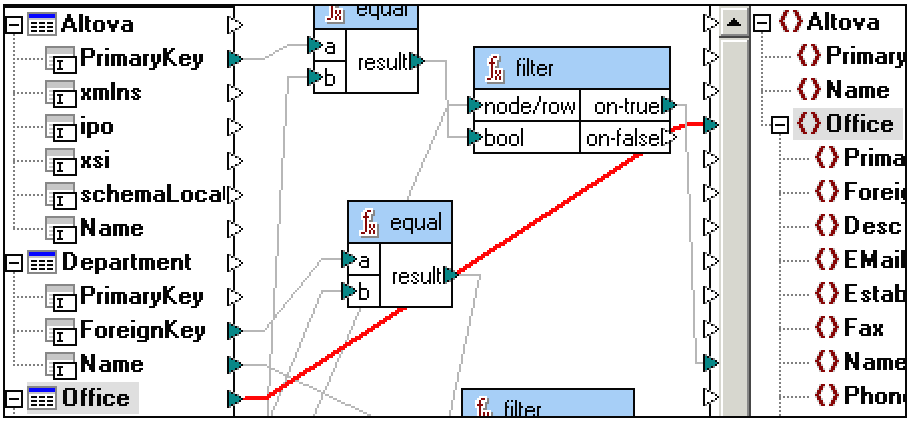
- Both office and department names are displayed. The **order** of their occurrence is currently determined by the structure of the target schema/document.



```
<?xml version="1.0" encoding="UTF-8"?>
<Altova xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Name>Nanonull Partners, Inc.</Name>
    <Department>
      <Name>Administration</Name>
      <Name>Marketing</Name>
      <Name>Engineering</Name>
      <Name>IT &amp; Technical Support</Name>
      <Name>Administration</Name>
      <Name>Marketing</Name>
      <Name>IT &amp; Technical Support</Name>
    </Department>
  </Office>
</Altova>
```

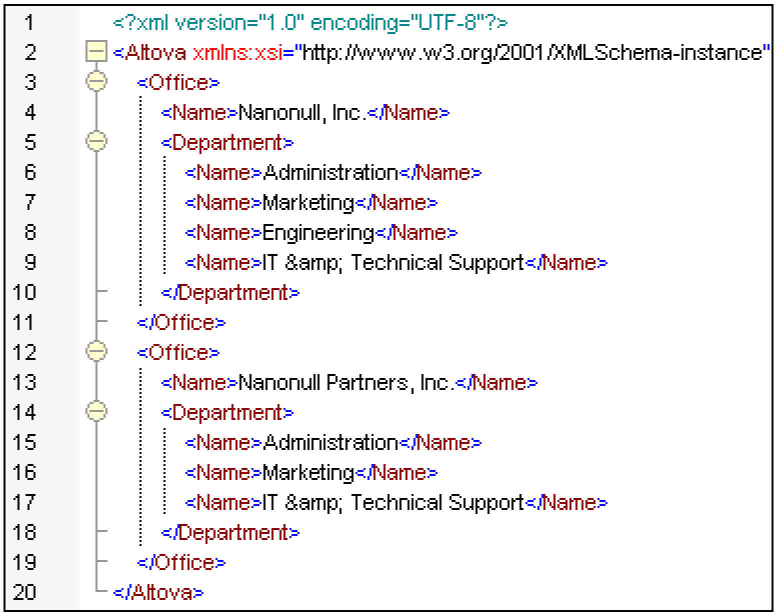
It would, of course, be more useful to output each office with its respective departments.

- To achieve this create a mapping **between** the Office items. Deactivate the "Auto complete child items" function by clicking the icon before doing this, if it is active.



The mapping now defines:

- "for each Office element, output the office name and then all departments in that office"



```
<?xml version="1.0" encoding="UTF-8"?>
<Altova xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Department>
      <Name>Administration</Name>
      <Name>Marketing</Name>
      <Name>Engineering</Name>
      <Name>IT &amp; Technical Support</Name>
    </Department>
  </Office>
  <Office>
    <Name>Nanonull Partners, Inc.</Name>
    <Department>
      <Name>Administration</Name>
      <Name>Marketing</Name>
      <Name>IT &amp; Technical Support</Name>
    </Department>
  </Office>
</Altova>
```

7.7 Mapping large databases with MapForce

When using large databases in mappings, MapForce creates all database relations between the imported tables, of the whole database. This is due to the fact that the application cannot automatically decide which tables will be used in the mapping process, all possibilities have to be covered. These type of large databases increase the size of the database component exponentially, which can lead to performance or memory issues.

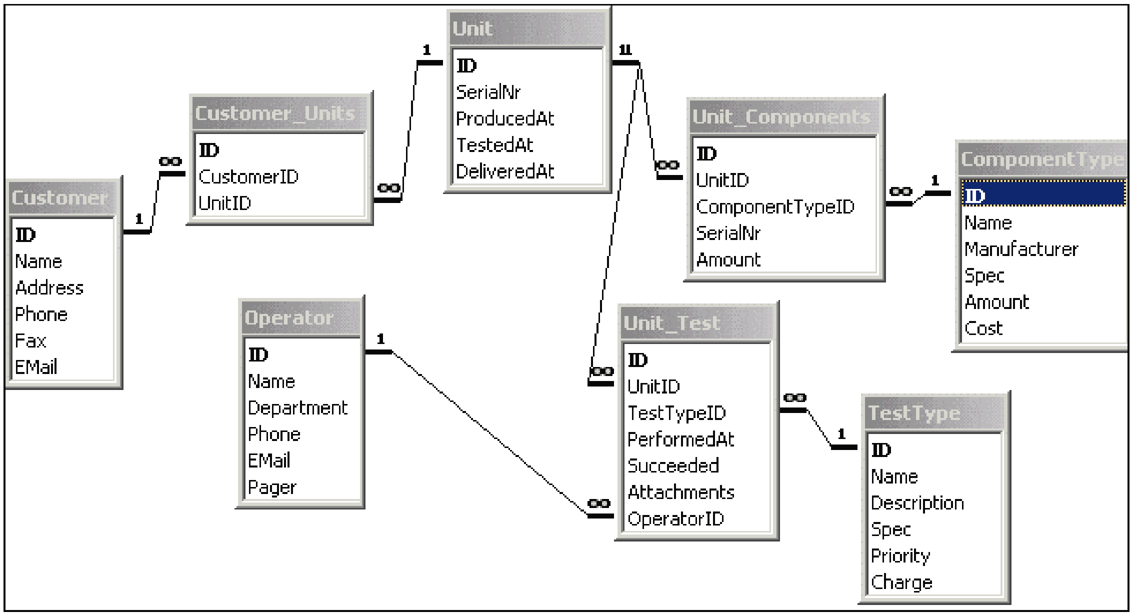
The solution to this dilemma is to create multiple database components, of the same database, which only use/import those tables that are needed for the mapping process. This method also makes for a more intuitive mapping.

E.g.
In a production company, various components are assembled to produce customer defined units. Before delivery, the units undergo a unit test and all results are stored in a database.

At some point during the prototype testing phase, it is discovered that a batch of components are faulty, and a recall has to be initiated. The goal of the mapping is to generate a list of all affected customers to whom a letter must be sent.

In this case the mapping defines:
For the ComponentType name = "Prototype" AND the Manufacturer = "Noname",
Select all related Customers and their requisite details.

The relationship diagram of the example database discussed in this section, is shown below:

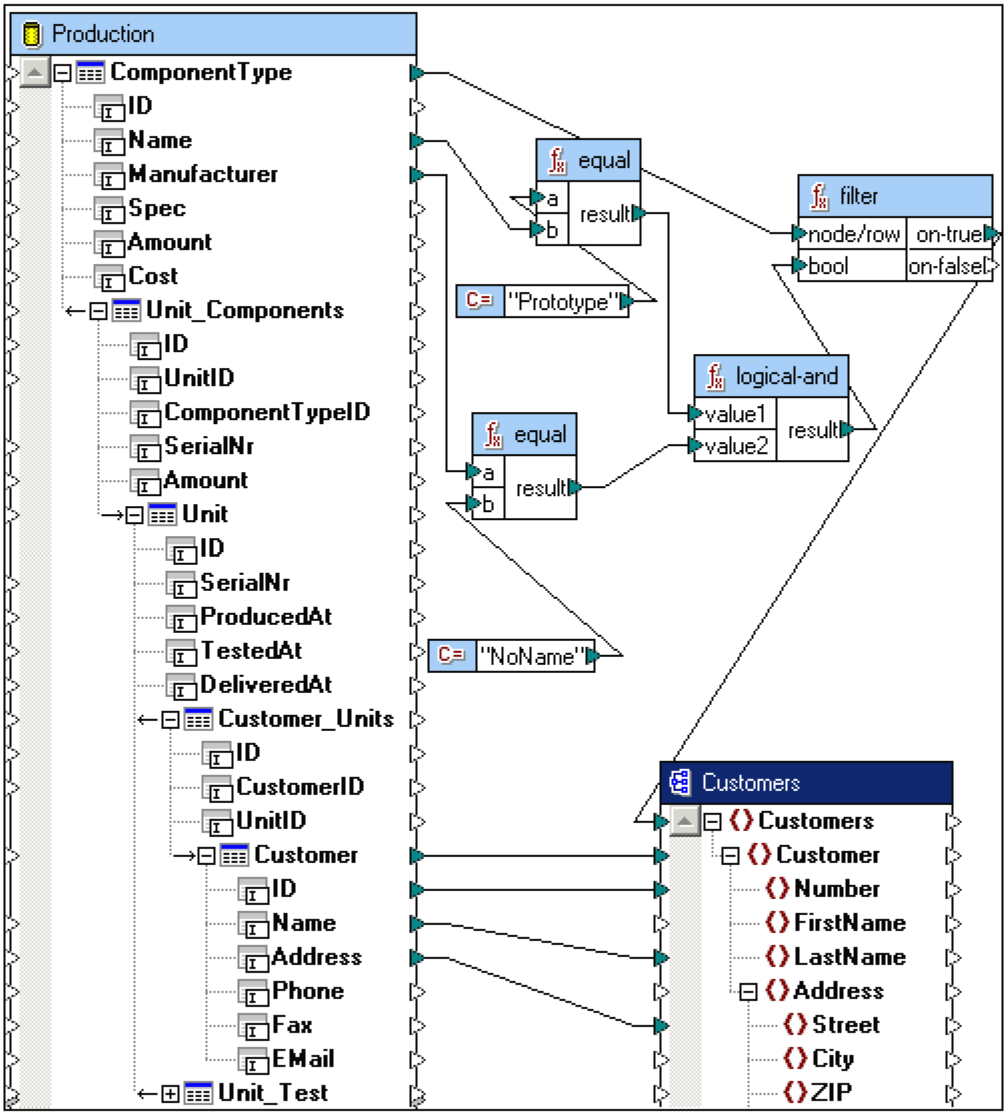


7.7.1 Complete database import

Option 1:
Import the complete database i.e. with **all** the tables it contains. The Product database component therefore contains all tables, with each table appearing as a "root" table along with all its related tables.

Using the ComponentType table as the root table: the mappings filter out:

- the Component Name "Prototype" AND
- the Manufacturer "NoName", along with
- the related Customer ID and address data



Only approximately one tenth of the database relations/hierarchy is used in this instance, the other 90% are redundant and cause a large overhead.

7.7.2 Partial database import

Option 2:
Import only those tables that are necessary to extract the necessary information i.e.:

- retrieve all defective units
- retrieve all customers to whom these units were supplied

Insert two database components, from the same database, importing different sets of tables

Component 1, insert the following tables:

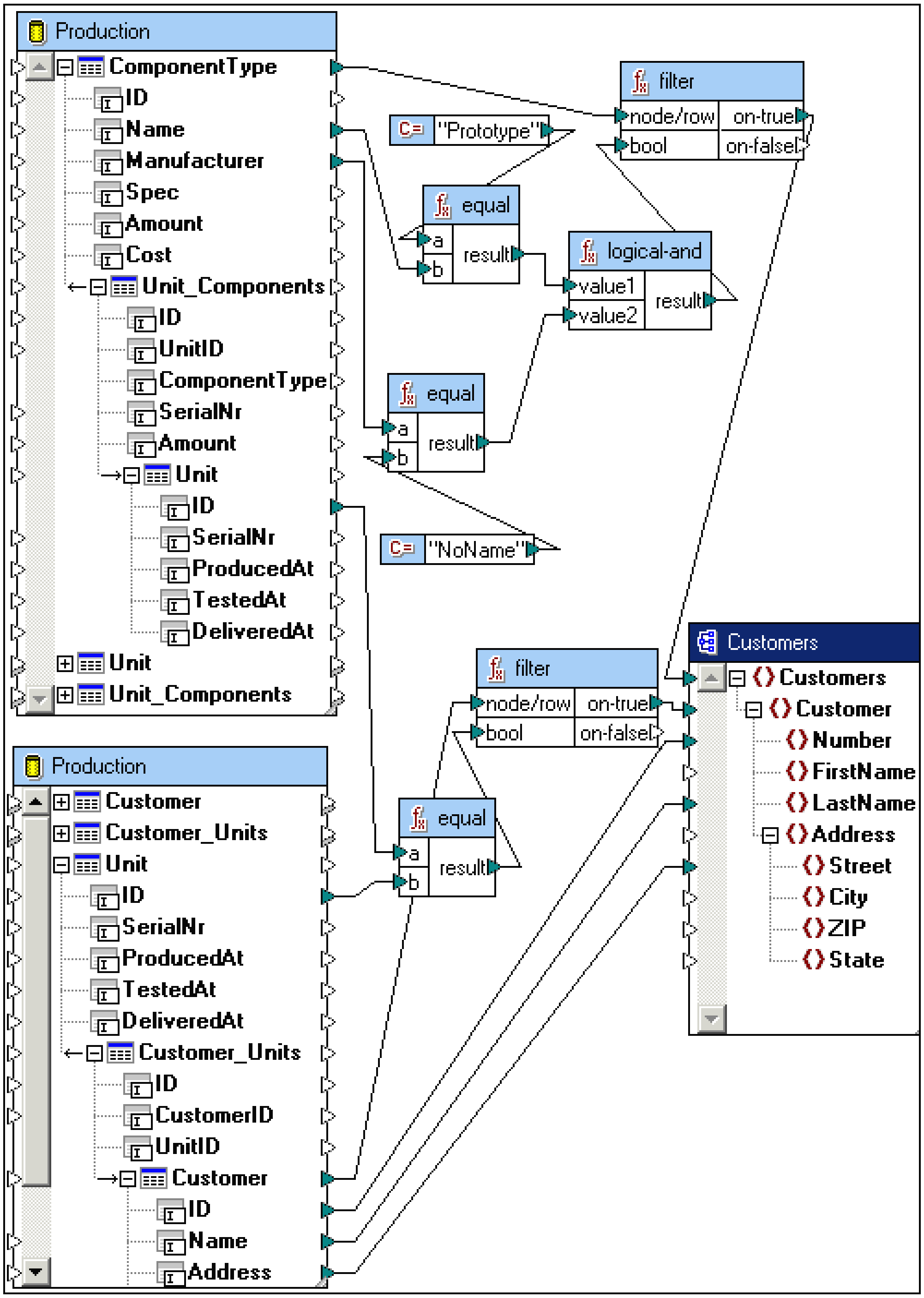
- ComponentType
- Unit_Components
- Unit

Component 2, insert:

- Unit
- Customer_Units
- Customer

Mapping process:

- filter out the Component Name "Prototype" AND the Manufacturer "NoName" (component 1)
- use the "equal" function compare the unit ID from component 1 with the unit ID from component 2
- if the IDs are equal, use the filter component to pass on the associated customer data from component 2 to the Customers XML file.



Please note:

Try to restrict the number of tables used in a single database component to about 10 related tables. This will ensure speedy loading and processing, as well as make the mapping process more intuitive.

7.8 Database filters and queries

When generating program code, MapForce optimizes database access by generating direct database queries where possible. The MAPFORCE **filter** component in conjunction with specific functions, is what makes this possible.

The filter component **generally** retrieves every record of a specific table and checks each to see if the filter condition is satisfied. If it is, the record is forwarded to the on-true/on-false parameters. This generates a select statement something like: **select "type" from "expense-item"**. This method is time consuming when using large databases, and an alternative method is used which transfers the workload to the database.

MapForce analyzes the mapping and checks for specific functions that support direct queries. Select statements are then generated for these functions, e.g. **select * from "expense-item" where type = "Travel".** Most of the work is now done by the database and the resulting dataset is then passed on for further processing.

The MapForce functions that support direct queries are show below.

Operators available for all database types:

MapForce function	**Database function**
"equal"	"="
"not-equal"	"<>"
"equal-or-greater"	">="
"equal-or-less"	"<="
"less"	"<"
"greater"	">"
"logical-or"	"or"
"logical-and"	"and"
"add"	"+"
"subtract"	"-"
"multiply"	"*"
"divide"	"/"
"modulus"	"%"

Functions for all database types:

"logical-not"	"not"

MS SQLServer specific functions:

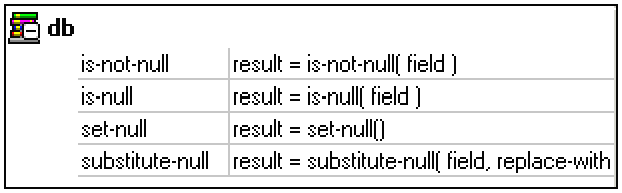
MapForce function	**Database function**
"floor"	"FLOOR()"
"ceiling"	"CEILING()"
"round"	"ROUND()"
"concat"	"+"
"substring"	"SUBSTRING()"
"contains"	"CHARINDEX()"
"string-length"	"LEN()"
"uppercase"	"UPPER()"
"lowercase"	"LOWER()"
"find-substring"	"CHARINDEX()"
"empty"	"IsEmpty()"

MS Access specific functions:

MapForce function	**Database function**
"round"	"Round()"
"concat"	"+"
"substring"	"Mid()"
"contains"	"InStr(1,..)"
"string-length"	"Len()"
"uppercase"	"UCase()"
"lowercase	"LCase()"
"find-substring"	"InStr(1,..)"
"empty"	"IsEmpty()"

7.9 Database, null processing functions

New null processing functions have been added to the DB language library.



db

is-not-null	result = is-not-null(field)
is-null	result = is-null(field)
set-null	result = set-null()
substitute-null	result = substitute-null(field, replace-with

is-not-null
Returns false if the field is null, otherwise returns true.

is-null
Returns true if the field is null, otherwise returns false.

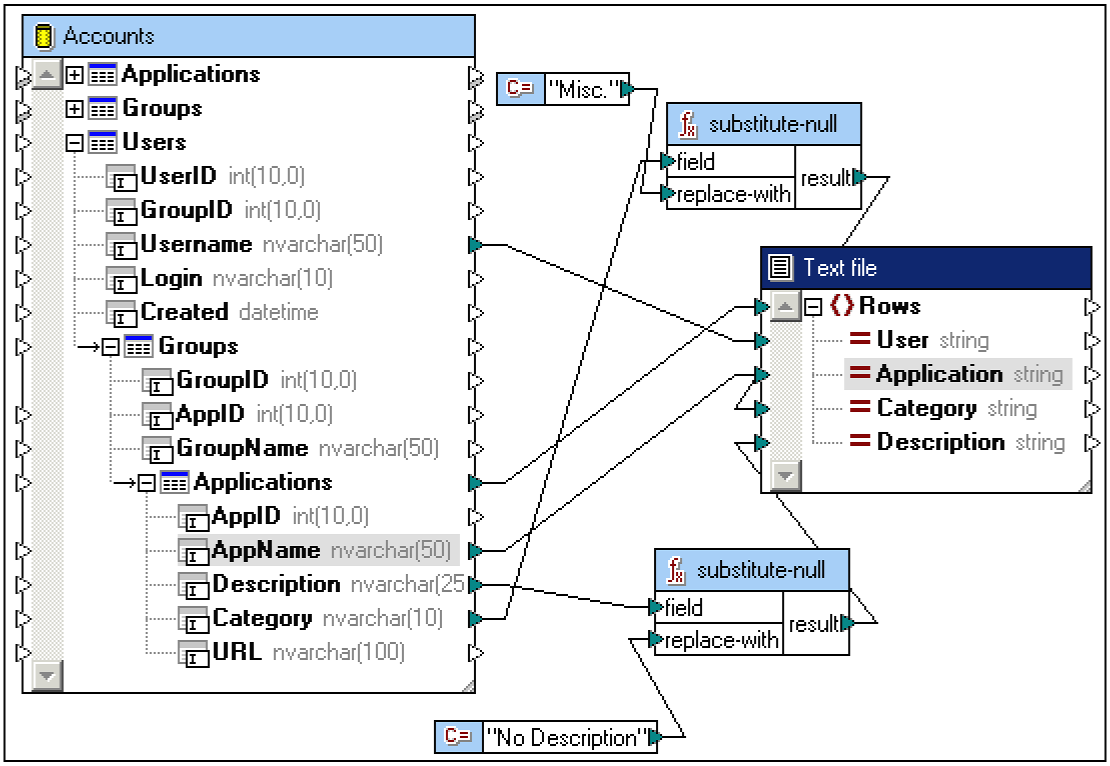
set-null
Used to set a database column, or text field to null. This function will also overwrite a default value with null.

Please note:

- Connecting this function to another function will generally not lead to a null result! (The null input will be cast to "", 0, or "false".)
- Connecting to special functions, Filters and IF-Else conditions works as expected, fields are set to null.
- Using set-null as an input for a simpleType element will not create that element in the target component.
- Connecting this function to a complexType element, as well as a table, or row is not allowed. A validation error occurs when this is done.

substitute null
Used to map the current field content if it exists, otherwise use the item mapped to the replace-with parameter.

The image below shows an example of the substitute-null function in use, and is available as "**DB-ApplicationList**" in the ...\MapForceExamples folder.



The first function checks if a Category entry exists in the Applications table. As one does not exist for the Notepad application, "Misc" is mapped to the Category item of the Text file.

The second function checks if a Description entry exist, and maps the string "No description" if one does not exist, which is also the case with the Notepad application.

Chapter 8

MapForce, CSV and Text files

8 MapForce, CSV and Text files

MapForce now includes support for the mapping of flat file formats, i.e. CSV files and Text files as both source and target components. Please note that you need to select one of the programming languages (Java, C#, or C++) as the mapping output, to be able to work with text, or CSV files.

This bi-directional mapping support includes:

- XML schema to/from flat file formats
- Database to/from flat file formats
- There are two ways that mapped flat file data can be generated/saved:
- By clicking the Output tab which generates a preview using the built-in MapForce engine, selecting the menu option **Output | Save output file**, or clicking the 💾 icon, to save the result
- By selecting **File | Generate code in | Java, C#, or C++** then compiling and executing the generated code.

Please note:

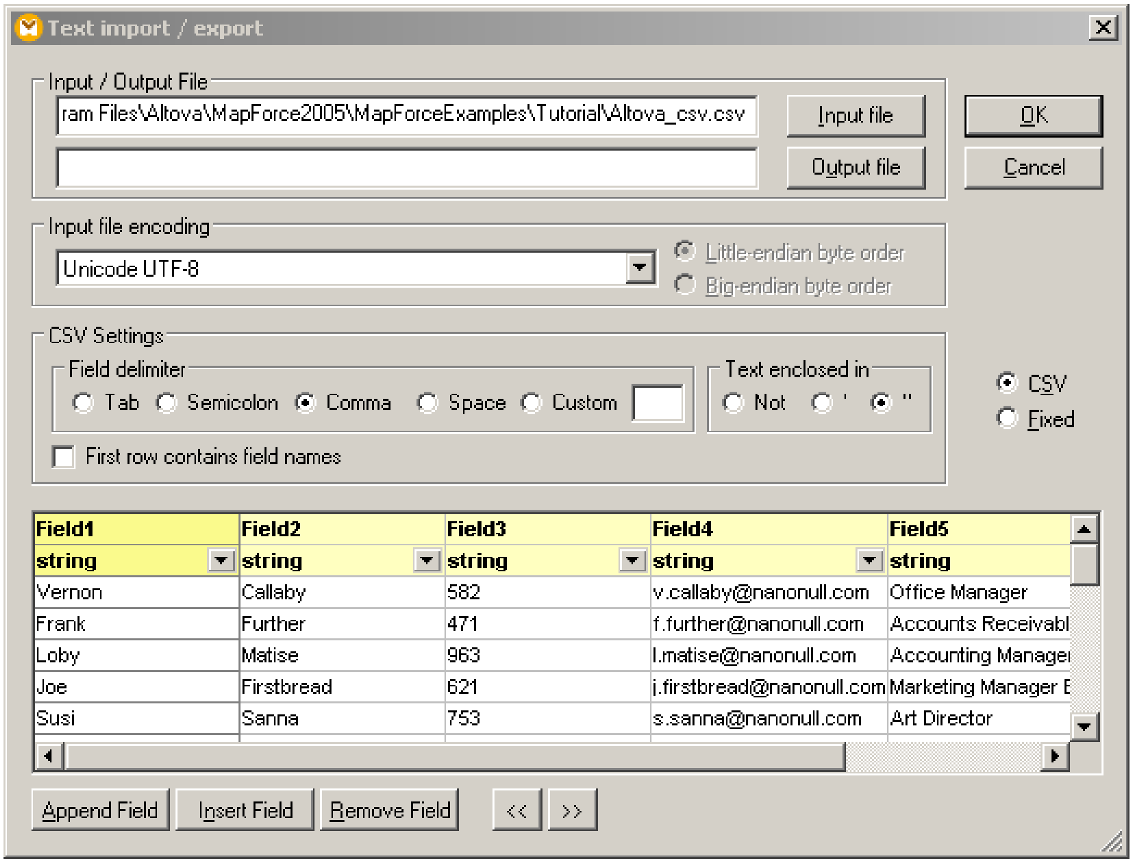
All the following examples using CSV files as source or target components, can also be accomplished with Fixed length text files. The only difference is that the field lengths have to be defined manually, please see "Mapping Fixed Length Text files" on how define field lengths.

8.1 Mapping CSV files to XML

This example maps a simple CSV file to an XML file, based on the MFCompany.xsd schema file. All the files used in the following examples are available in the **...\MapForceExamples\Tutorial** folder.

- Having made sure you selected one of the programming languages, **Java**, **C#**, or **C++**, by clicking the respective toolbar icon.

1. Select the menu option **Insert | Text file**, or click the "Insert Text file" icon
This opens the Text import / export dialog box, in which you can select the type of file you want to work with CSV, or Fixed length files. The CSV radio button is active by default.



Input / Output File
ram Files\Altova\MapForce2005\MapForceExamples\Tutorial\Altova_csv.csv

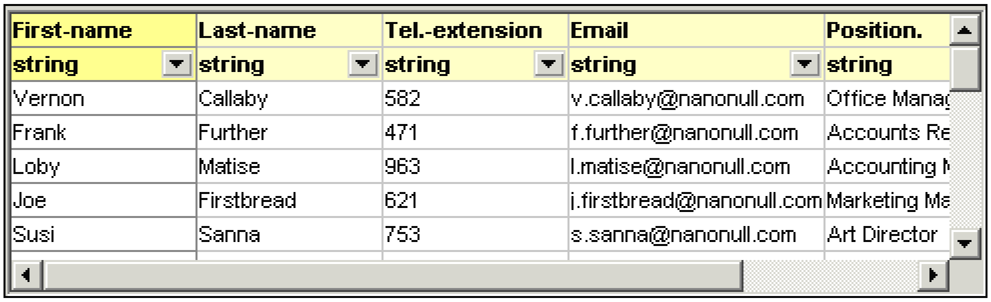
Input file encoding
Unicode UTF-8
● Little-endian byte order
○ Big-endian byte order

CSV Settings
Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom
Text enclosed in: ○ Not ○ ' ● "
□ First row contains field names

● CSV
○ Fixed

Field1	Field2	Field3	Field4	Field5
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager
Frank	Further	471	f.further@nanonull.com	Accounts Receivabl
Loby	Matise	963	l.matise@nanonull.com	Accounting Manager
Joe	Firstbread	621	j.firstbread@nanonull.com	Marketing Manager E
Susi	Sanna	753	s.sanna@nanonull.com	Art Director

2. Click the **Input file** button and select the CSV file, e.g. Altova_csv.csv. The file contents are now visible in the Preview window. Please note that the Preview window only displays the first 20 rows of the text file.
3. Click into the Field1 header and change the text, e.g. First-name. Do the same for all the other fields, e.g. Last-name, Tel.-extension, Email, and Position.

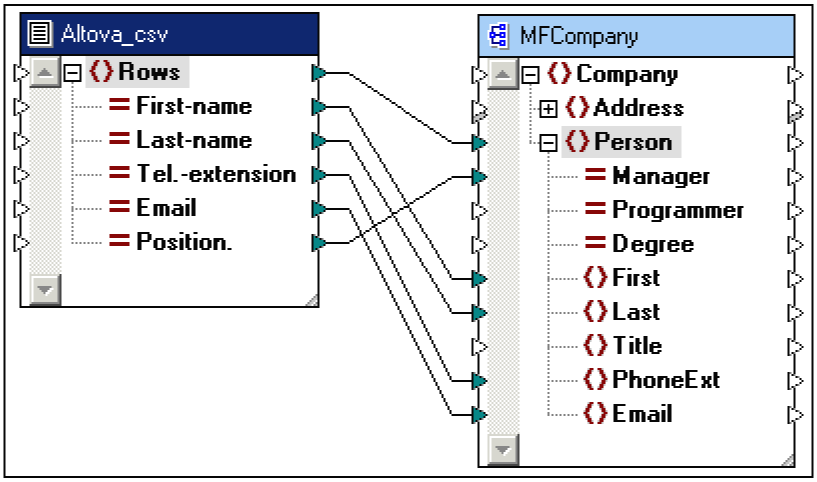


First-name	Last-name	Tel.-extension	Email	Position.
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manag
Frank	Further	471	f.further@nanonull.com	Accounts Re
Loby	Matise	963	l.matise@nanonull.com	Accounting M
Joe	Firstbread	621	j.firstbread@nanonull.com	Marketing Ma
Susi	Sanna	753	s.sanna@nanonull.com	Art Director

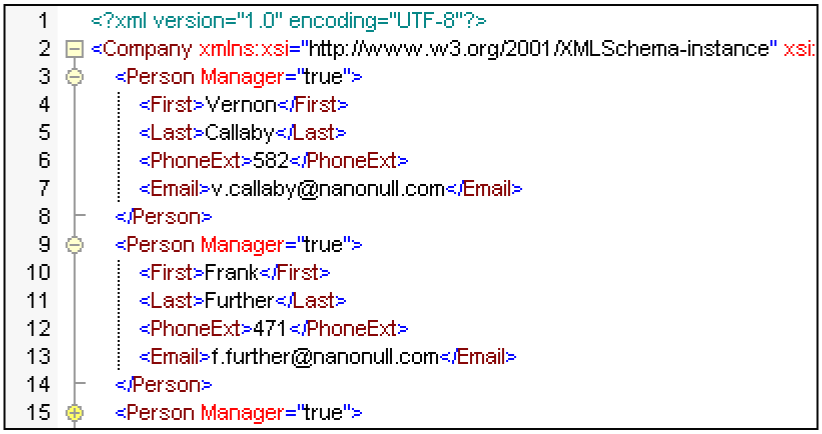
Please note:

Hitting the **Tab** keyboard key, allows you to cycle through all the fields: header1, field type1, header2 etc.

3. Click the OK button when you are satisfied with the settings.
   The CSV component is now visible in the Mapping.
4. Select the menu option **Insert | XML/Schema file** and select **MFCompany.xsd**.
5. Click No, when asked if you want to supply a sample XML file, and select Company as the root element.



6. Map the corresponding items of both components, making sure to map the **Rows** item to the **Person** item in the schema target, then click the Output tab to see the result.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  <Person Manager="true">
    <First>Vernon</First>
    <Last>Callaby</Last>
    <PhoneExt>582</PhoneExt>
    <Email>v.callaby@nanonull.com</Email>
  </Person>
  <Person Manager="true">
    <First>Frank</First>
    <Last>Further</Last>
    <PhoneExt>471</PhoneExt>
    <Email>f.further@nanonull.com</Email>
  </Person>
  <Person Manager="true">
```

The data from the CSV file have been successfully mapped to an XML file.

Please note:

The connector from the **Rows** item in the CSV file, to the Person item in the schema is essential, as it defines which elements will be iterated through; i.e. for each Row in the CSV file a new Person element will be created in the XML output file.

Please see the examples that follow, on how the **Rows** item influences the output if you are mapping **to** a CSV, or fixed length text file.

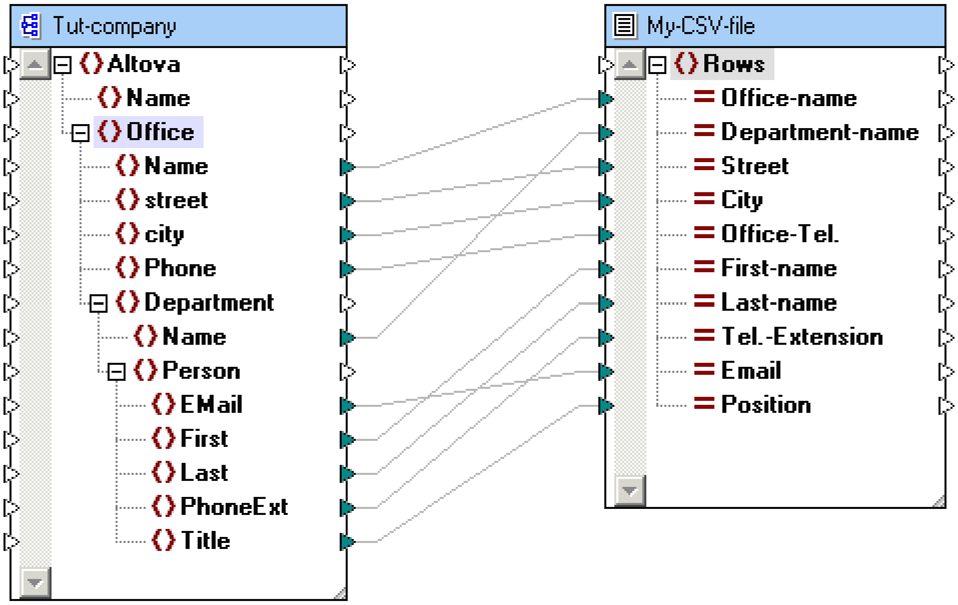
8.2 Mapping XML to CSV, or fixed length text files

This example is available in the ...\MapForceExamples\Tutorial folder as **Tut-xml2csv.mfd**.

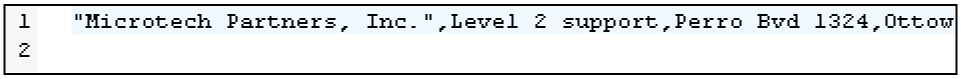
- **Tut-company.xsd** and **Tut-company.xml** are the source schema and XML data source respectively.
- "My-CSV-file" is the text file component. The name is entered in the "Input file" field of the Text import /export dialog box.

  The mapping example is for illustration purposes only, it is not supposed to be a real-life example.

The diagram below shows how you would generally expect to map an XML file to a CSV file.



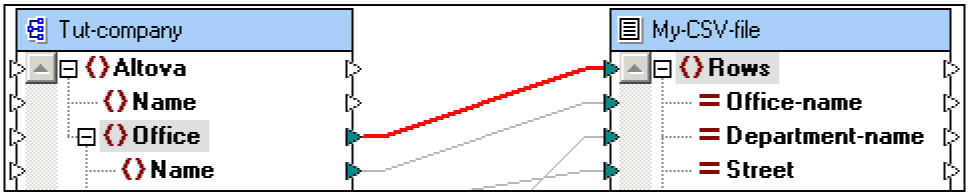
Clicking the Output tab produces the result you see below, which may not be what you expect, we only see output for the first office.



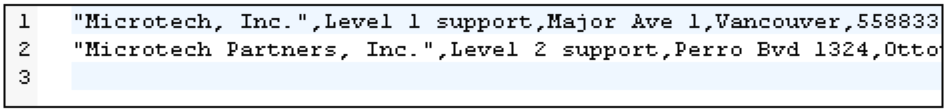
```
1   "Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Ottow
2
```

In order to be able to iterate through all offices and have the output appear in the CSV file, it is necessary to connect Office to Rows. What this means is: for each Office item of the source XML, create a Row in the target CSV file. MapForce allows you to specify the field, or item which is to act as the "root"/iterator for the output using the **Rows** item.

Mapping the **Office** item to the **Rows** item, results in all individual Offices (and mapped items) being output.

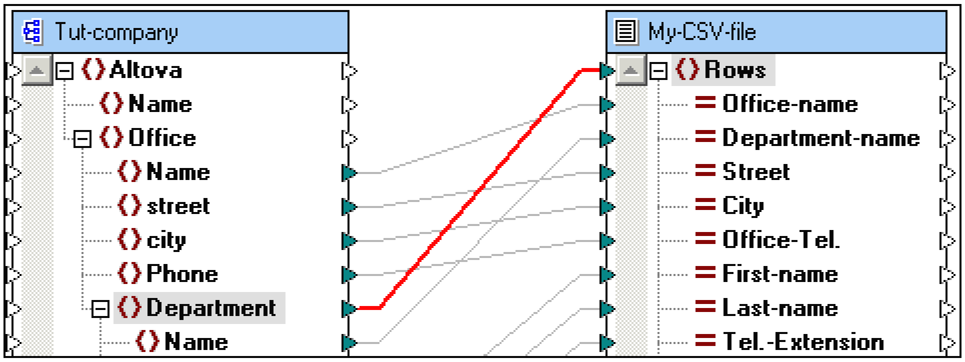


The Office items are output in the source file sequence.

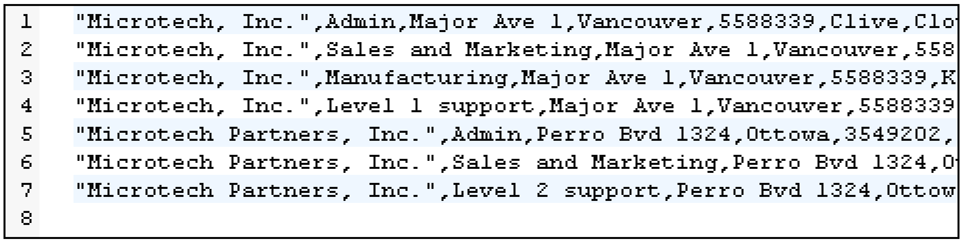


```
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,558833
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Otto
```

Mapping **Department** to the **Rows** item results in all of the Departments being output.

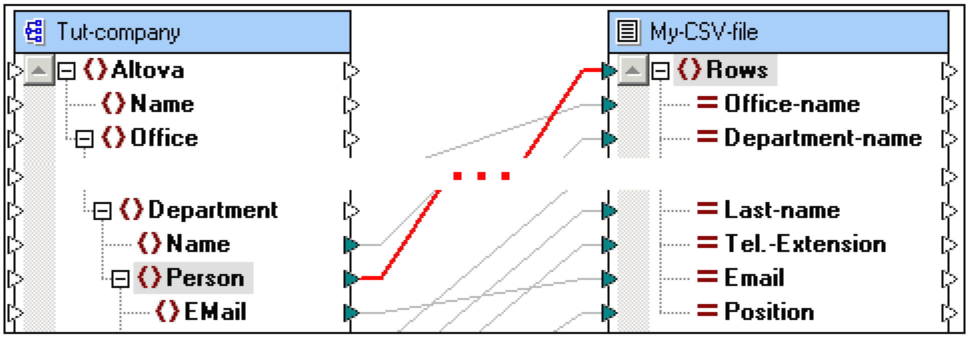


The Departments are output in the source file sequence, for each Office.

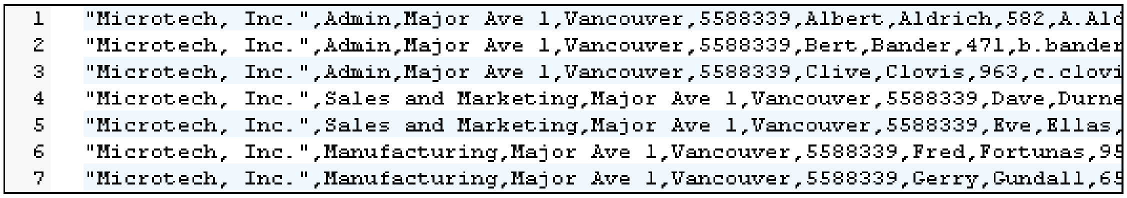


```
"Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clo
"Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,558
"Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,K
"Microtech, Inc.",Level 1 support,Major Ave 1,Vancouver,5588339
"Microtech Partners, Inc.",Admin,Perro Bvd 1324,Ottowa,3549202,
"Microtech Partners, Inc.",Sales and Marketing,Perro Bvd 1324,O
"Microtech Partners, Inc.",Level 2 support,Perro Bvd 1324,Ottow
```

Mapping **Person** to the **Rows** item results in all the Persons being output.



The Persons are output in the source file sequence, i.e. each Person within each Department, for each Office.



```
"Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Albert,Aldrich,582,A.Ald
"Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Bert,Bander,471,b.bander
"Microtech, Inc.",Admin,Major Ave 1,Vancouver,5588339,Clive,Clovis,963,c.clovi
"Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,5588339,Dave,Durne
"Microtech, Inc.",Sales and Marketing,Major Ave 1,Vancouver,5588339,Eve,Ellas,
"Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,Fred,Fortunas,95
"Microtech, Inc.",Manufacturing,Major Ave 1,Vancouver,5588339,Gerry,Gundall,65
```

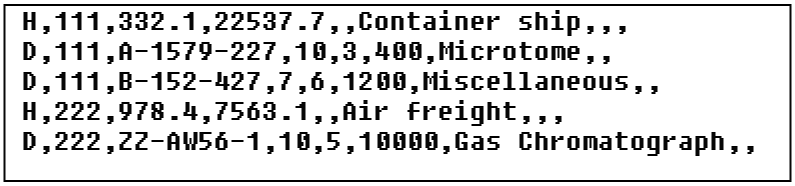
8.3 Creating hierarchies from CSV and fixed length text files

This example is available in the ...\MapForceExamples\Tutorial folder as **Tut-headerDetail.mfd** .

The example uses a CSV file with fields that define the specific record types, and has the following format:

- Field 1: H defines a header record and D a detail record.
- Field 2: A common/key for both header and detail records.
- Each header/detail record is on a separate line.

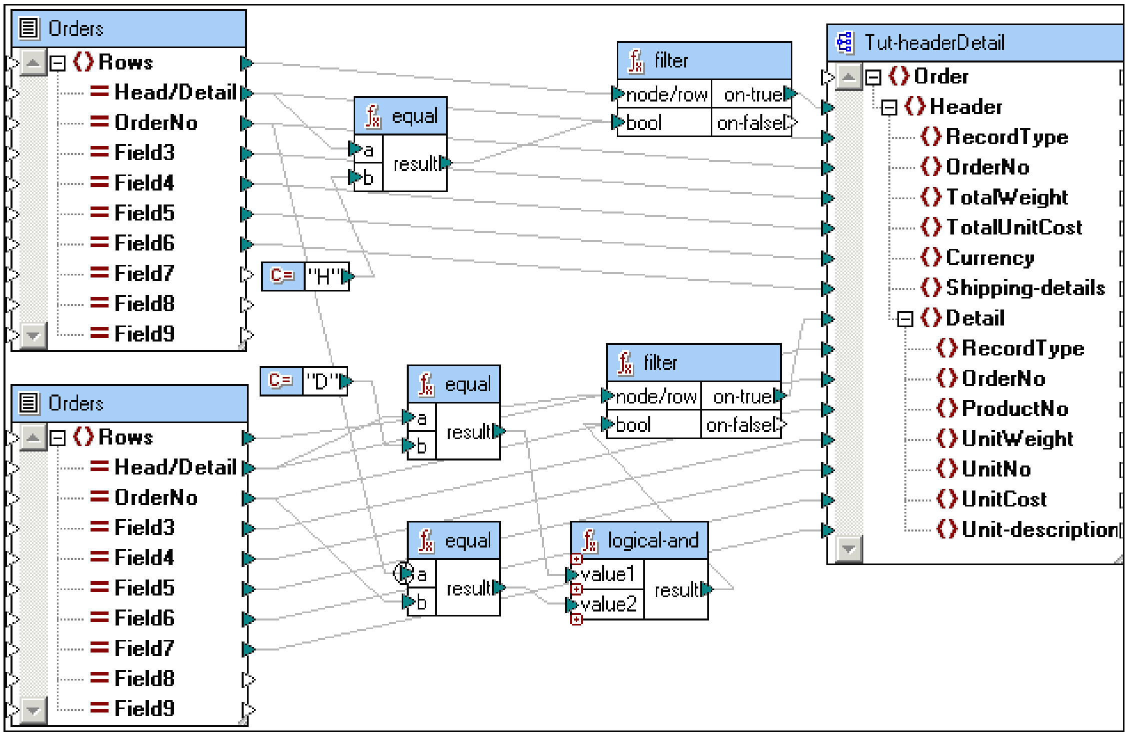
Creating hierarchical XML structures from flat files using "Key" fields
The contents of the Orders.csv file are shown below.



```
H,111,332.1,22537.7,,Container ship,,,
D,111,A-1579-227,10,3,400,Microtome,,
D,111,B-152-427,7,6,1200,Miscellaneous,,
H,222,978.4,7563.1,,Air freight,,,
D,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

Aim of the mapping is to:

- Map the flat file CSV to an hierarchical XML file, and
- Filter out the Header records, designated with an H, and
- Associate the respective detail records, designated with a D, with each of the header records



For this to be achieved the header and detail records must have one common field. In this case the common field, or key, is the second field of the CSV file, i.e. **OrderNo**. In the CSV file both the first header record and the following two detail records, contain the common value 111.

Notes on the mapping:
The Orders.csv file has been inserted twice to make the mapping more intuitive.

The **Tut-headerDetail.xsd** schema file has a hierarchical structure: Order is the root element, with Header as its child element, and Detail being a child element of Header.

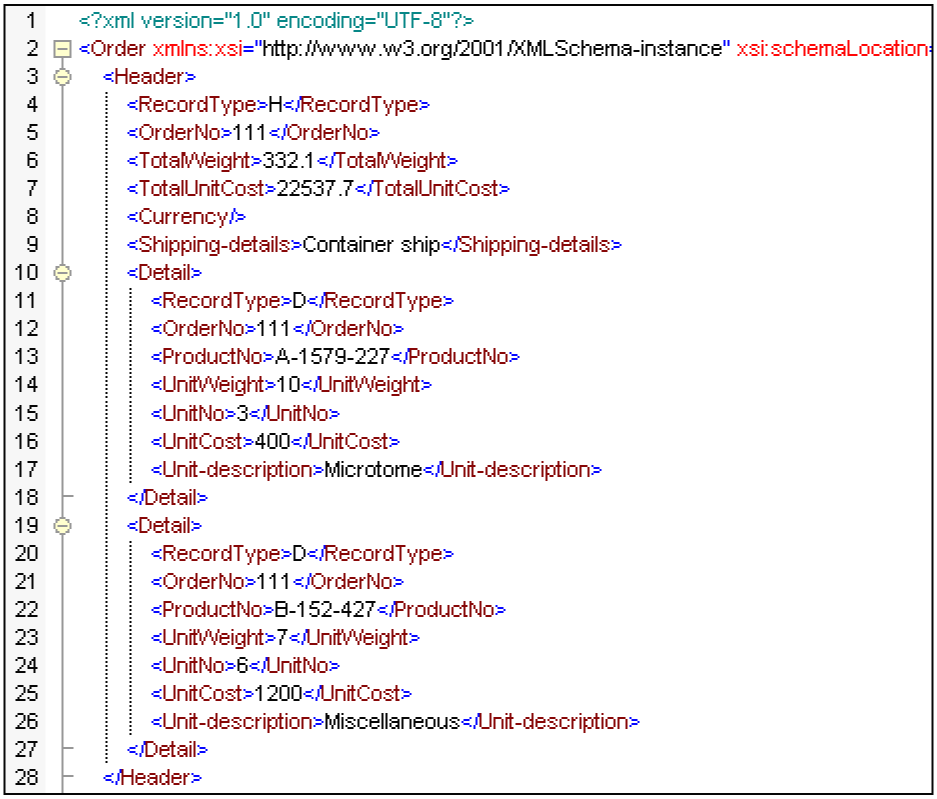
The first Orders.csv file supplies the **Header** records (and all mapped fields) to the Header item in the schema target file. The filter component is used to filter out the H records. The **Rows** item supplies these filtered records to the Header item in the schema file.

The second Orders.csv file supplies the **Detail** records (and all mapped fields) by filtering out the Detail records that match the OrderNo key of the Header record. This is achieved by:

- Comparing the **OrderNo** field of the Header record with the same field of the Detail records, using the **equal** function (the priority context is set on the **a** parameter for enhanced performance).
- Using the **Logical-and** function to only supply those Detail records containing the same OrderNo field, as the Header record.

  The **Rows** item supplies these filtered records to the Header and Detail items in the schema file, through the on-true parameter of the filter function.

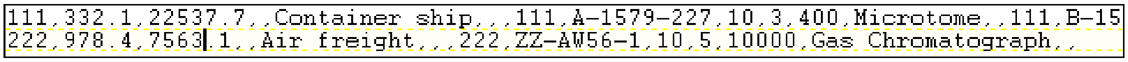
Clicking the Output tab produces the XML file displayed below. Each Header record contains its data, and all associated Detail records that have the same Order No.



```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
  <Header>
    <RecordType>H</RecordType>
    <OrderNo>111</OrderNo>
    <TotalWeight>332.1</TotalWeight>
    <TotalUnitCost>22537.7</TotalUnitCost>
    <Currency/>
    <Shipping-details>Container ship</Shipping-details>
    <Detail>
      <RecordType>D</RecordType>
      <OrderNo>111</OrderNo>
      <ProductNo>A-1579-227</ProductNo>
      <UnitWeight>10</UnitWeight>
      <UnitNo>3</UnitNo>
      <UnitCost>400</UnitCost>
      <Unit-description>Microtome</Unit-description>
    </Detail>
    <Detail>
      <RecordType>D</RecordType>
      <OrderNo>111</OrderNo>
      <ProductNo>B-152-427</ProductNo>
      <UnitWeight>7</UnitWeight>
      <UnitNo>6</UnitNo>
      <UnitCost>1200</UnitCost>
      <Unit-description>Miscellaneous</Unit-description>
    </Detail>
  </Header>
```

The second example uses a slightly different CSV file and is available in the ...\MapForceExamples\Tutorial folder as **Head-detail-inline.mfd**. however:

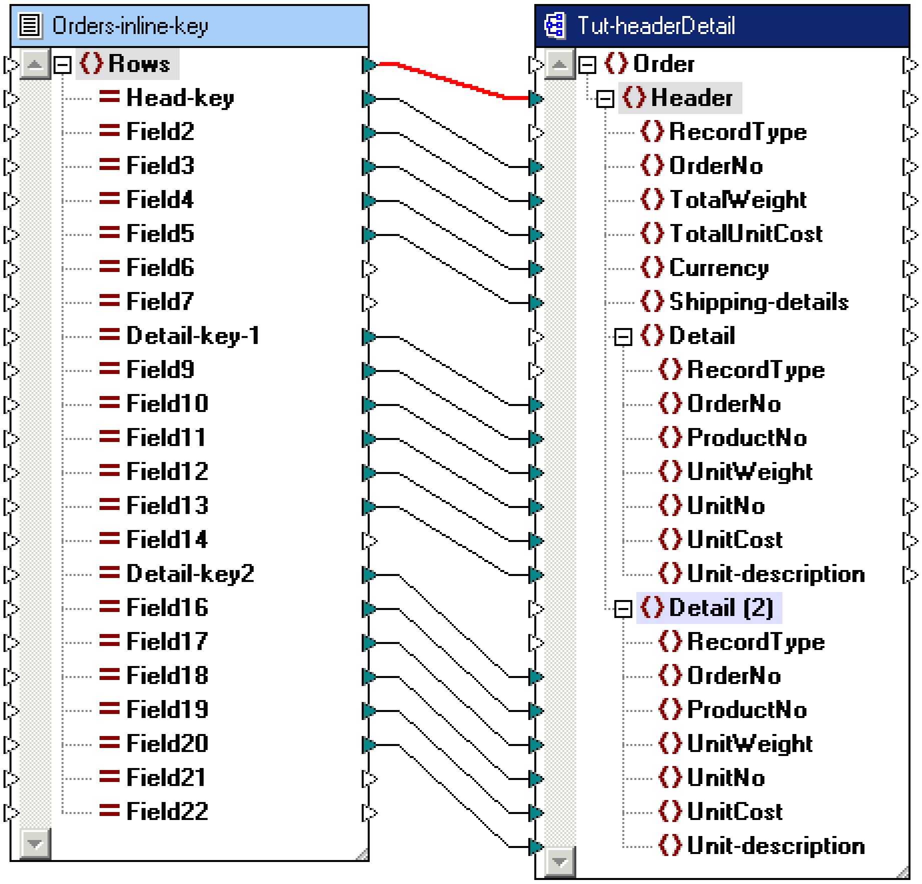
- No record designator (H, or D) is available
- A common/key field, the first field of the CSV file, still exists for both header and detail records (Head-key, Detail-key...). The field is mapped to OrderNo in the schema target
- Header and all respective Detail fields are all on the same line.



```
111,332.1,22537.7,,Container ship,,,111,A-1579-227,10,3,400,Microtome,,111,B-15
222,978.4,7563.1,,Air freight,,,222,ZZ-AW56-1,10,5,10000,Gas Chromatograph,,
```

Please note:

- The key fields are mapped to the respective OrderNo items in the schema target.
- The Detail item in the schema target file has been duplicated, and is displayed as **Detail (2)**. This allows you to map the second set of detail records to the correct item.
- The result of this mapping is exactly the same XML file that was produced in the above example.

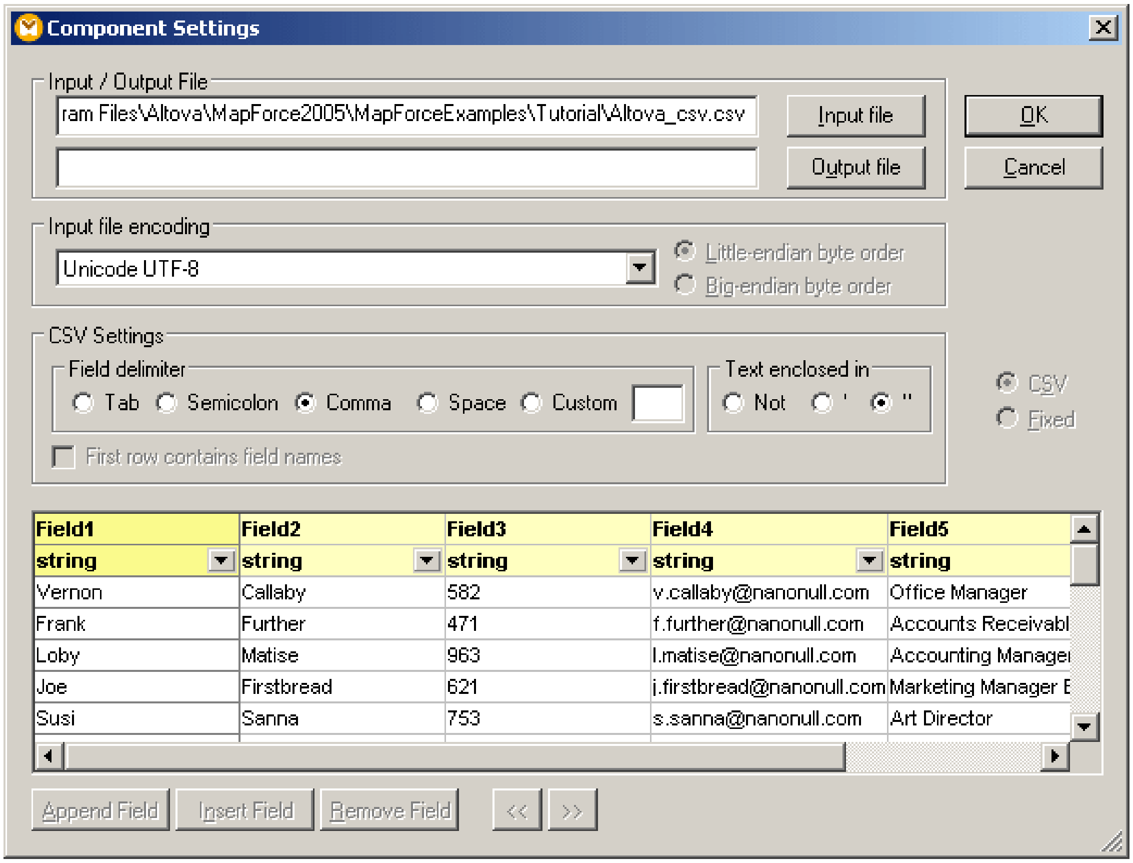


8.4 CSV file options

Right click the **Altova_csv** component and select **Component Settings** to open the dialog box.

CSV Text import / export options:
When defining field formats in this dialog box, type checking of the respective fields is automatically performed. If the input data and the field format defined here to do not agree, then the data is highlighted in red. E.g. changing field2 from string to integer would make all surnames of that column appear in red.

Please note:
The field types that one can select for a specific column, are based on the default XML schema datatypes. E.g. The Date type is in the form: YYYY-MM-DD.



Input / Output File: ram Files\Altova\MapForce2005\MapForceExamples\Tutorial\Altova_csv.csv

Input file encoding: Unicode UTF-8 ● Little-endian byte order ○ Big-endian byte order

CSV Settings
Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom
Text enclosed in: ○ Not ○ ' ● "
☐ First row contains field names

● CSV ○ Fixed

Field1	Field2	Field3	Field4	Field5
string	string	string	string	string
Vernon	Callaby	582	v.callaby@nanonull.com	Office Manager
Frank	Further	471	f.further@nanonull.com	Accounts Receivabl
Loby	Matise	963	l.matise@nanonull.com	Accounting Manage
Joe	Firstbread	621	j.firstbread@nanonull.com	Marketing Manager E
Susi	Sanna	753	s.sanna@nanonull.com	Art Director

Input file:
Select the CSV file you want to use as the source file for this component.

Please note:

This field can remain empty if you are using the Text file component as a **target** for another text, XML file etc. In this case, the file encoding automatically defaults to UTF-8. You can define the field type, field names, formatting etc. and click OK to create a text file target.

Clicking the **Output** tab then allows you to **save** this text file, by clicking the "Save generated output as..." icon including its mapped contents.

Entering a name in this text box (without using a file extension) assigns this name to the component.

Output file:
Select the target file you want to output data to, when **generating code** with MapForce. Make sure that the input and output files are different when generating code, or the source file will be overwritten when the code is executed. This option is only used when generating code for Java, C++, or C#.

File encoding:
Allows you to define/select the encoding of the input text file. If there is no entry in the Input file field, then the encoding automatically defaults to UTF-8.

CSV Settings - Field delimiter:
Select the delimiter type for the text file (CSV files are comma delimited "," per default). You can also enter a custom delimiter in the **Custom** field.

Click into the Custom field and:

- Hit a keyboard key to enter a new value, or
- Double click in the Custom field, to mark the current value, and hit a different keyboard key to change the entry.

First row contains field names:
Sets the **values** in the first record of the text file as the column headers (visible in the preview window). The column headers then appear as the item names when the Text component is displayed in the mapping.

Text enclosed in:
Text files exported from legacy systems sometimes enclose text values in quotes to distinguish them from numeric values.

Select this option if the text file contains strings which include the Field delimiter that you have currently defined. The same delimiter character can then occur within a string without affecting the text file segmentation/partitioning. E.g. your fields (strings) contain a comma character "," but you are also using this character as the default CSV delimiter.

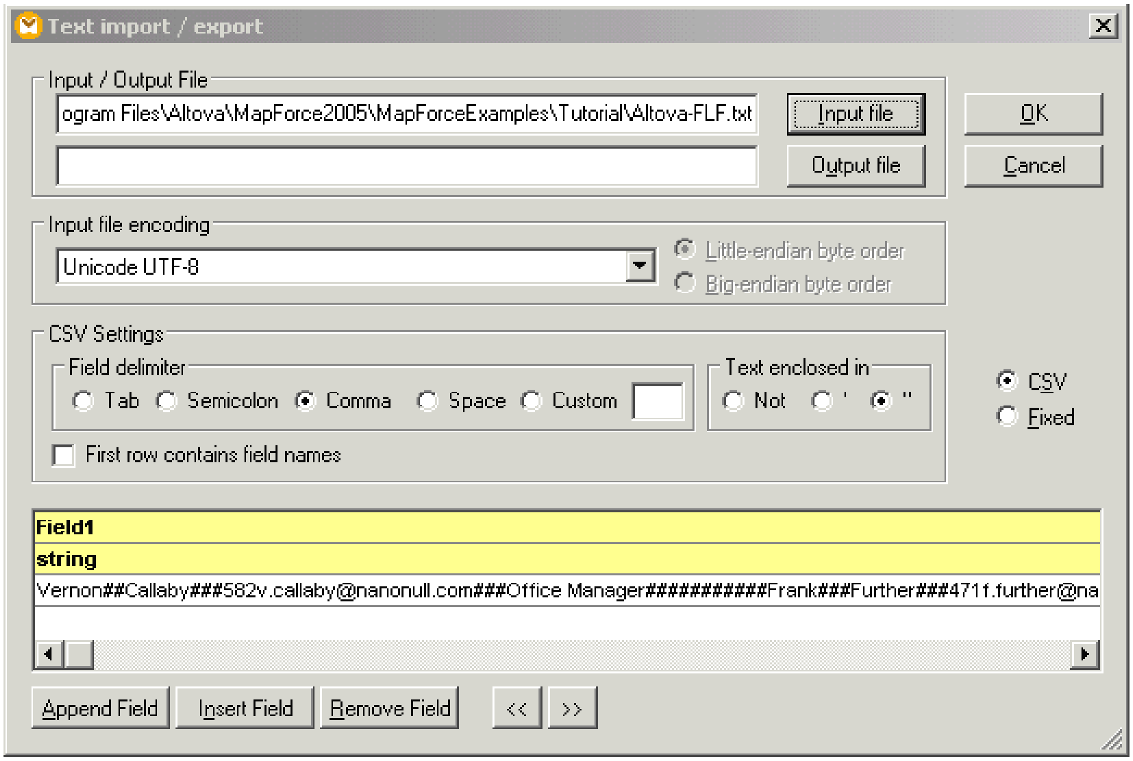
Append field, **Insert** field, **Remove** field:
Allows you to append, insert or remove fields in the preview window, which defines the structure of the CSV file.

Next / Previous
Clicking one of these buttons moves the currently active column left or right in the preview window.

8.5 Mapping Fixed Length Text files (to a database)

This example maps a simple text file to a MS Access database. The source text file is one continuous string with no carriage returns, or line feeds. All the files used in the following examples are available in the **...\MapForceExamples\Tutorial** folder.

1. Select the menu option **Insert | Text file**, or click the insert Text file icon .
   This opens the Text import / export dialog box, in which you can select the type of file, and specific settings, you want to work with.
2. Click the **Input file** button and select the **Altova-FLF.txt** file.
   You will notice that the file is made up of a single string, and contains fill characters of type #.



Input / Output File: ogram Files\Altova\MapForce2005\MapForceExamples\Tutorial\Altova-FLF.txt

Input file encoding: Unicode UTF-8 ● Little-endian byte order ○ Big-endian byte order

CSV Settings

Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom

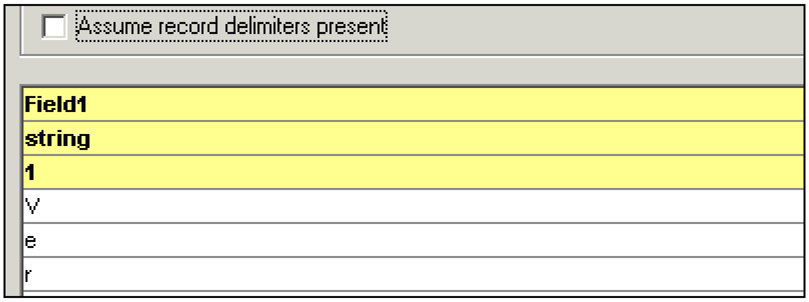
Text enclosed in: ○ Not ○ ' ● "

□ First row contains field names

● CSV
○ Fixed

Field1
string
Vernon##Callaby###582v.callaby@nanonull.com###Office Manager###########Frank###Further###471f.further@na

3. Click the **Fixed** radio button (below CSV).
4. Uncheck the "**Assume record delimiters present**" check box.



□ Assume record delimiters present

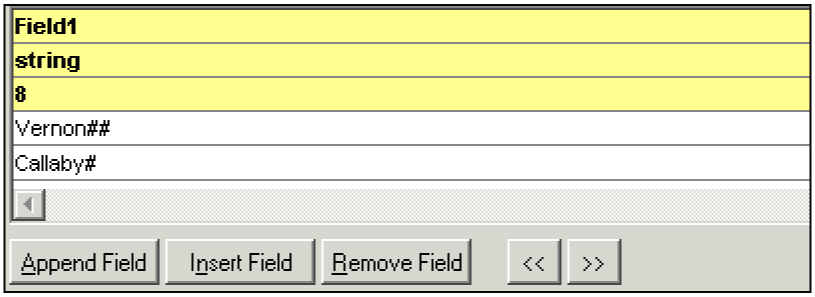
Field1
string
1
V
e
r

The preview changes at this point. What we now have, is a fixed format comprising of:

- a single field called "Field1"
- where the format is of type "string", and the
- field length is one character (V from person Vernon)

Field 1 now contains additional data.

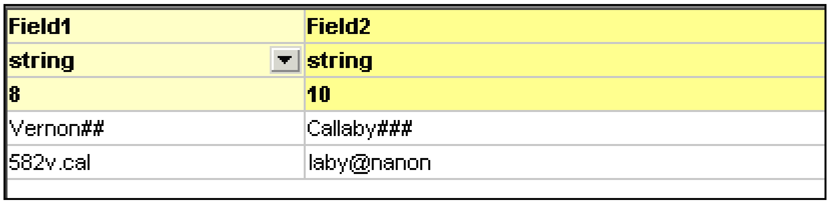
5. Click into the row containing the **1** character, change the value to 8, and hit Return.



Field1
string
8
Vernon##
Callaby#

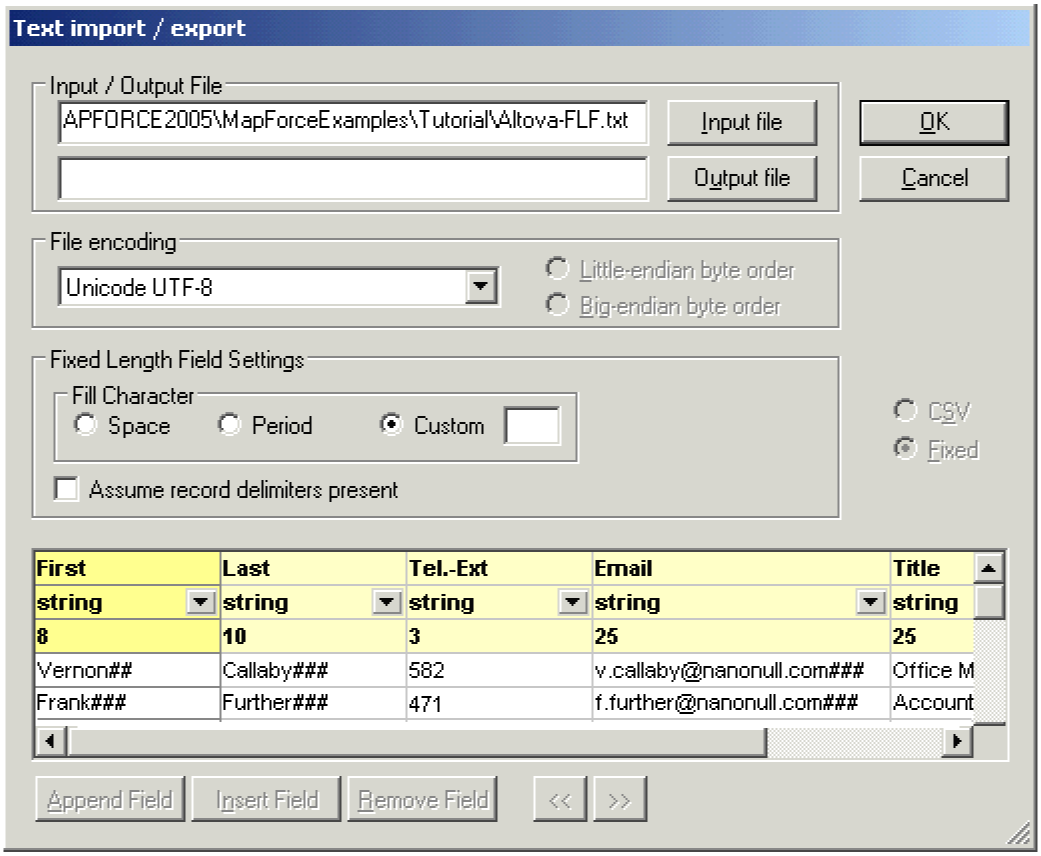
More data is now visible in the first column, which is now defined as 8 characters wide.

6. Click the **Append Field** button to add a new field, and make the length of the second field, 10 characters.



Field1	Field2
string	string
8	10
Vernon##	Callaby###
582v.cal	laby@nanon

7. Use the same method to create three more fields of the following lengths: 3, 25, and 25 characters, and change the field headers to make them easier to map: First, Last, Tel.-Ext, Email, Title. The preview will then look like this:



Input / Output File: APFORCE2005\MapForceExamples\Tutorial\Altova-FLF.txt

File encoding: Unicode UTF-8

Fixed Length Field Settings

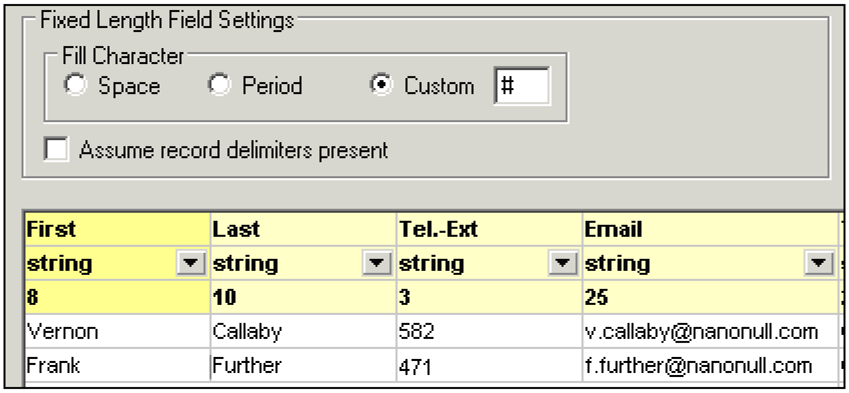
Fill Character: ○ Space ○ Period ● Custom

□ Assume record delimiters present

○ CSV ● Fixed

First	Last	Tel.-Ext	Email	Title
string	string	string	string	string
8	10	3	25	25
Vernon##	Callaby###	582	v.callaby@nanonull.com###	Office M
Frank###	Further###	471	f.further@nanonull.com###	Account

8. Click into the Custom text box of the Fixed Length Field Settings group, and enter the hash (#) character. This has the effect of removing the identical fill character from the text file being input.



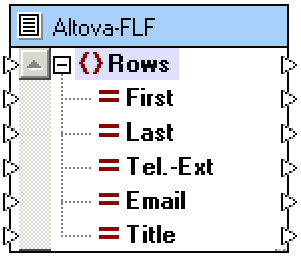
Fixed Length Field Settings

Fill Character: ○ Space ○ Period ● Custom #

□ Assume record delimiters present

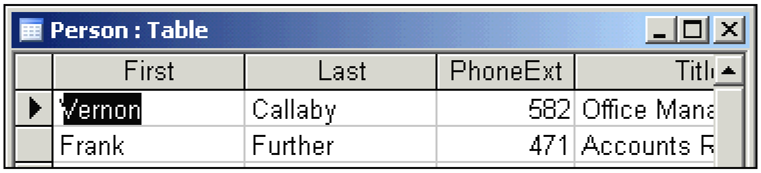
First	Last	Tel.-Ext	Email
string	string	string	string
8	10	3	25
Vernon	Callaby	582	v.callaby@nanonull.com
Frank	Further	471	f.further@nanonull.com

9. Click OK to complete the definition.



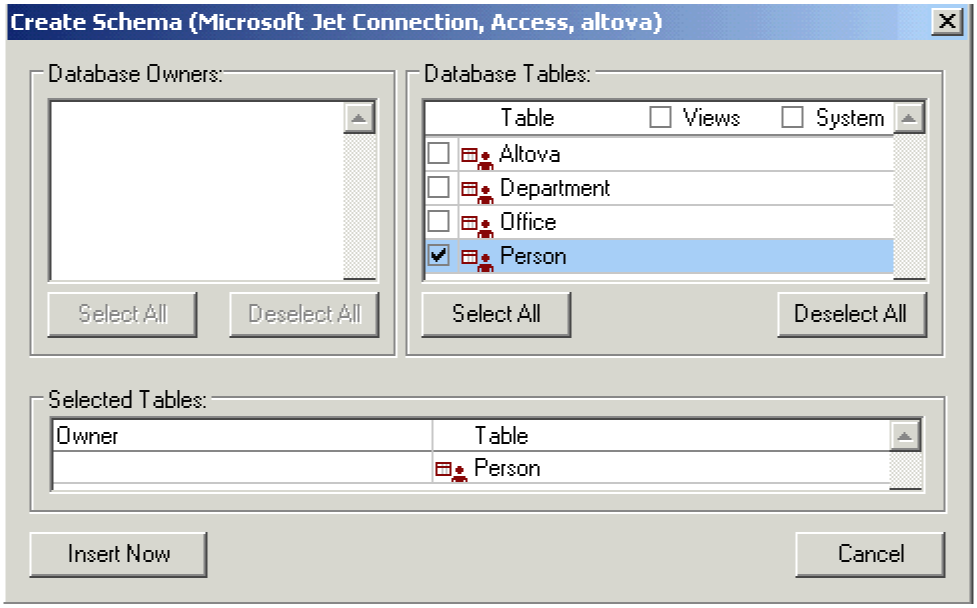
The Text file component appears in the Mapping window. Data can now be mapped to, and from, this component.

Mapping text files to a database:
This section uses the fixed length text file to update the Telephone extension entries in the **altova.mdb** database.


Person : Table

First	Last	PhoneExt	Titl
Vernon	Callaby	582	Office Mana
Frank	Further	471	Accounts R

1. Select the menu option **Insert | Database,** click the Microsoft Access radio button, then click Next.
2. Select the **altova.mdb** database available in the ...\MapForceExamples\Tutorial folder, and click Next.
3. Select the **Person** table by clicking the corresponding check box in the Database Tables list box.


Create Schema (Microsoft Jet Connection, Access, altova)

Database Owners:

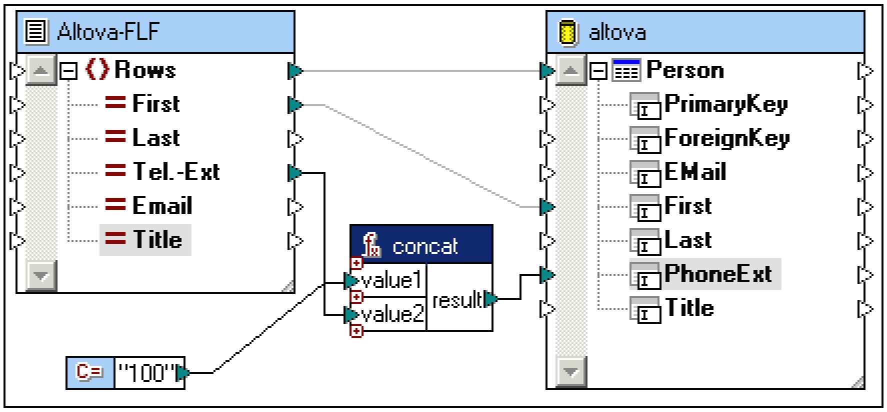
Database Tables:

Table	☐ Views	☐ System
☐ Altova		
☐ Department		
☐ Office		
☑ Person		

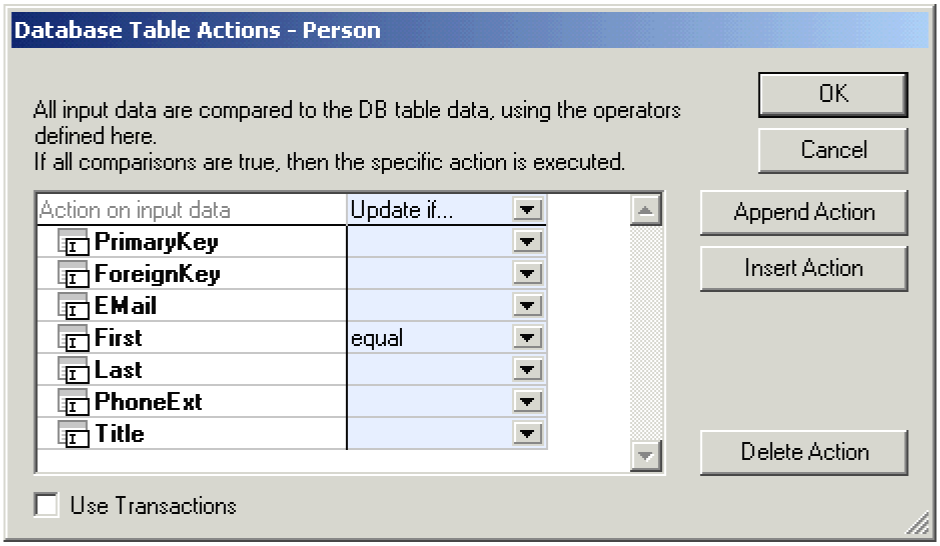
Selected Tables:

Owner	Table
	Person

4. Click the **Insert Now** button to create the database component.
5. Click the expand icon to see the table contents.
6. Drag the **concat** function from the libraries window into the Design tab.
7. Select the menu option **Insert | Constant**, click the Number radio button, and enter 100 as the new telephone extension prefix.
8. Create the mapping as shown in the graphic below.



9. Right click the Person entry and select Database table actions.


Database Table Actions - Person

All input data are compared to the DB table data, using the operators defined here.
If all comparisons are true, then the specific action is executed.

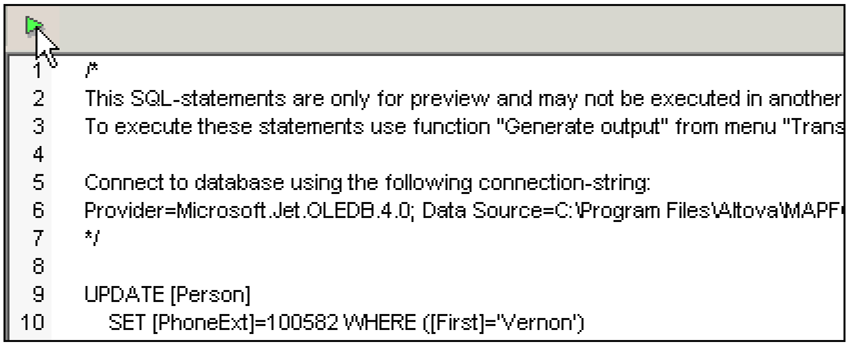
Action on input data	Update if...
PrimaryKey	
ForeignKey	
EMail	
First	equal
Last	
PhoneExt	
Title	

☐ Use Transactions

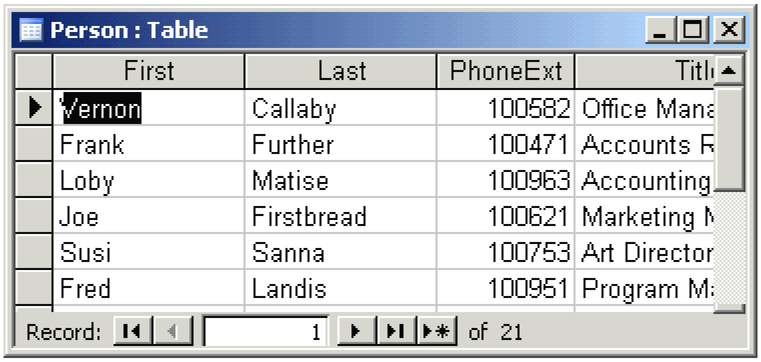
10. Click the "Action on input data" combo box and select the "Update if..." entry.
11. Click the combo box of the "First" row, select "equal", and click OK to confirm.

Person table data is only updated if the First names of the source and database field are identical. The action taken when this is true, is actually defined by the mapping. In this case the telephone extension is prefixed by 100, and placed in the PhoneExt field of the Person table.

12. Click the Output tab to generate the pseudo SQL statements, then click the Run SQL-script button to execute the SQL statements.


```
/*
This SQL-statements are only for preview and may not be executed in another
To execute these statements use function "Generate output" from menu "Trans

Connect to database using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MAPF
*/

UPDATE [Person]
   SET [PhoneExt]=100582 WHERE ([First]='Vernon')
```

The telephone extension fields of all persons are updated in the database.



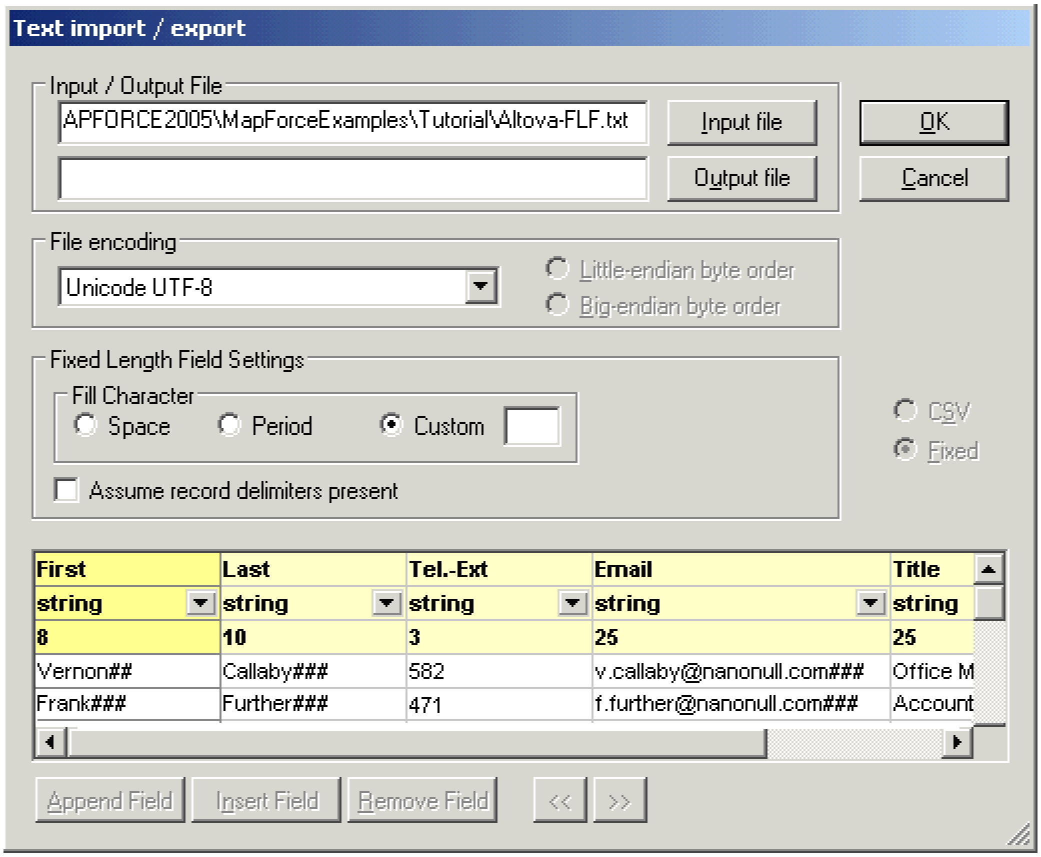
First	Last	PhoneExt	Titl
Vernon	Callaby	100582	Office Mana
Frank	Further	100471	Accounts R
Loby	Matise	100963	Accounting
Joe	Firstbread	100621	Marketing N
Susi	Sanna	100753	Art Director
Fred	Landis	100951	Program Ma

8.5.1 Fixed Length Text file options

Right click the **Altova-FLF** Text file component and select **Component Settings** to open the dialog box.

Fixed Length Text import / export options:
When defining field formats in this dialog box, type checking of the respective fields is automatically performed. If the input data and the field format defined here to do not agree, then the data is highlighted in red.

Please note:
The field types that one can select for a specific column, are based on the default XML schema datatypes. E.g. The Date type is in the form: YYYY-MM-DD.



Text import / export

Input / Output File
APFORCE2005\MapForceExamples\Tutorial\Altova-FLF.txt

File encoding
Unicode UTF-8
○ Little-endian byte order
○ Big-endian byte order

Fixed Length Field Settings
Fill Character
○ Space ○ Period ● Custom

□ Assume record delimiters present

○ CSV
● Fixed

First	Last	Tel.-Ext	Email	Title
string	string	string	string	string
8	10	3	25	25
Vernon##	Callaby###	582	v.callaby@nanonull.com###	Office M
Frank###	Further###	471	f.further@nanonull.com###	Account

Input file:
Select the text file you want to use as the source file for this component.

Please note:

This field can remain empty if you are using the Text file component as a **target** component for a mapping. In this case, the file encoding automatically defaults to UTF-8. You can define the field type, field names, formatting etc. and click OK to create a text file target.

Clicking the **Output** tab then allows you to **save** this text file, with its mapped output, by clicking the "Save generated output as..." icon .

Entering a name in this text box (without using a file extension) assigns this name to the component.

Output file
Select the target file you want to output data to, when **generating code** with MapForce. Make sure that the input and output files are different when generating code, or the source file will be overwritten when the code is executed. This option is only used when generating code for Java, C++, or C#.

File encoding
Allows you to define/select the encoding of the input text file. If there is no entry in the Input file field, then the encoding automatically defaults to UTF-8.

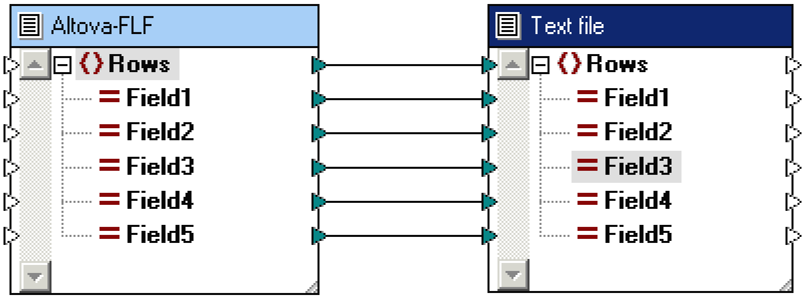
Fill Character
This option allows you to define the characters that are to be used to complete, or fill-in, the rest of the (fixed) field when the incoming data is less/shorter than the respective field definitions. The custom field allows you to define your own fill character in the Custom field.

Stripping fill characters:
If the incoming data already contains specific fill characters, and you enter the **same fill** character in the Custom field, then the incoming data will be stripped of those fill characters!

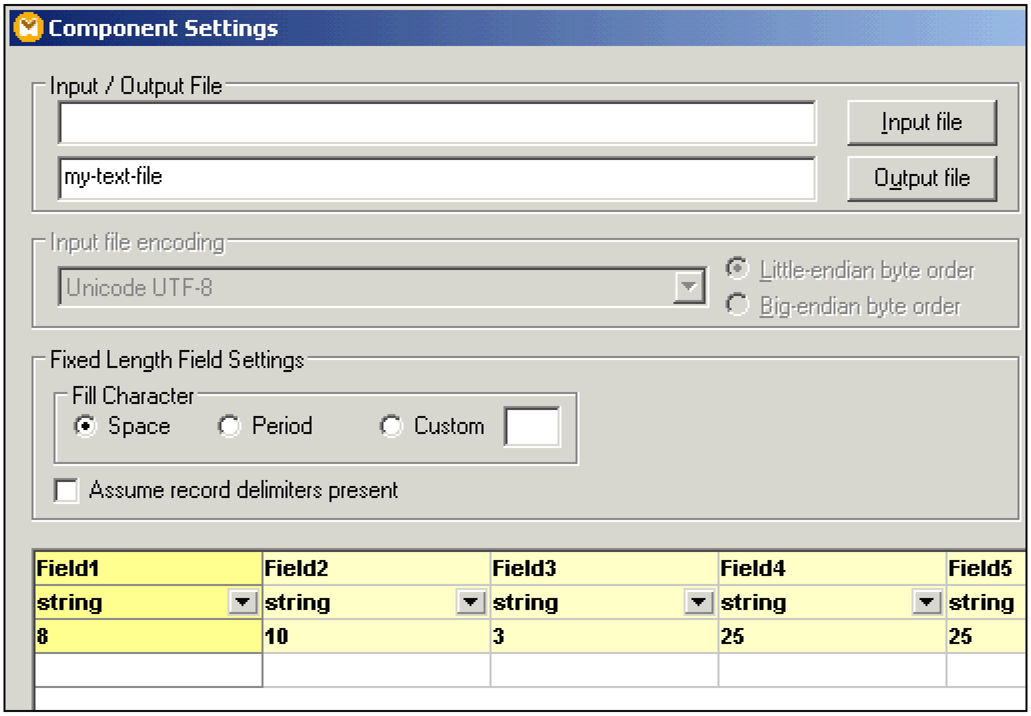
You can also enter a custom fill character in the **Custom** field. Click into the Custom field and:

- Hit a keyboard key to enter a new value, or
- Double click in the Custom field, to mark the current value, and hit a different keyboard key to change the entry.

Assume record delimiters present:
If a fixed length text file (single string) is the data source for another fixed length text file (mapping of two text files), then setting this option in the **target** file, creates new rows after the last column of the target has been filled.



In the example above the Altova-FLF text file is mapped to an empty target text file, my-text-file.



Component Settings

Input / Output File

my-text-file

Input file encoding: Unicode UTF-8

● Little-endian byte order
○ Big-endian byte order

Fixed Length Field Settings

Fill Character: ● Space ○ Period ○ Custom

□ Assume record delimiters present

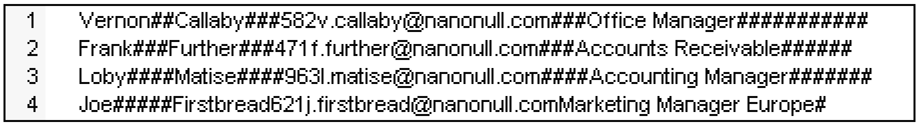
Field1	Field2	Field3	Field4	Field5
string	string	string	string	string
8	10	3	25	25

Please note:

- There is no **Input file** entry, which means that this text component only receives data from the mapped source component.
- Field lengths have been defined to correspond to the field lengths in the data source "Altova-FLF".
- No data can be seen in the preview, as the target component is not based on an existing text file.
- Clicking the Output tab, displays the mapped data.

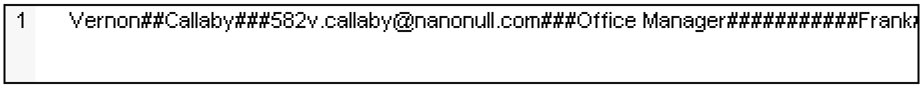
Check box "Assume record delimiters present"

- if checked,
  a new record is created after the sum of the defined field lengths, i.e. in this case all fields add up to 71 characters, a new record will be created for character 72.



```
1   Vernon##Callaby###582v.callaby@nanonull.com###Office Manager###########
2   Frank###Further###471f.further@nanonull.com###Accounts Receivable######
3   Loby####Matise####963l.matise@nanonull.com####Accounting Manager#######
4   Joe#####Firstbread621j.firstbread@nanonull.comMarketing Manager Europe#
```

- if unchecked,
  the mapped data appears as one long string, including the defined fill characters.



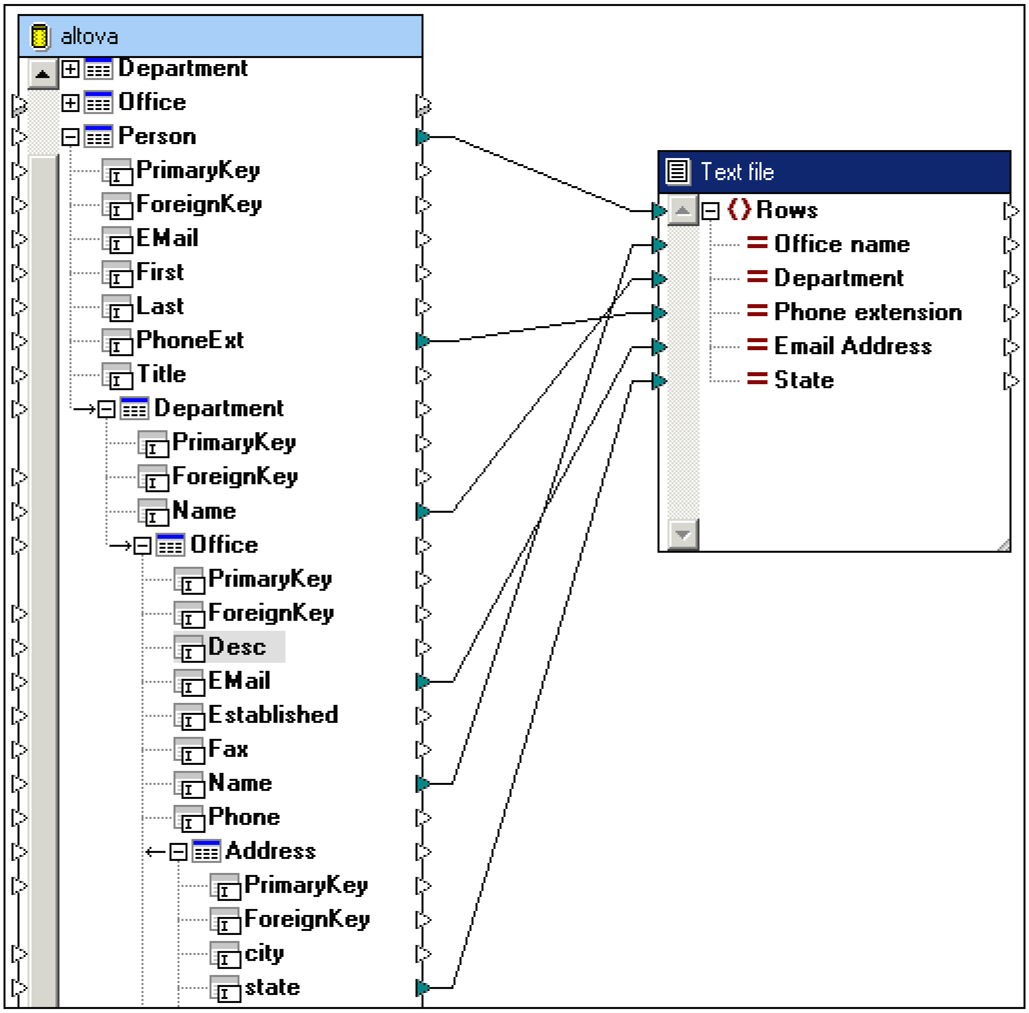
```
1   Vernon##Callaby###582v.callaby@nanonull.com###Office Manager###########Frank#
```

Append field, **Insert** field, **Remove** field:
Allows you to append, insert or remove fields in the preview window, which defines the structure of the CSV file.

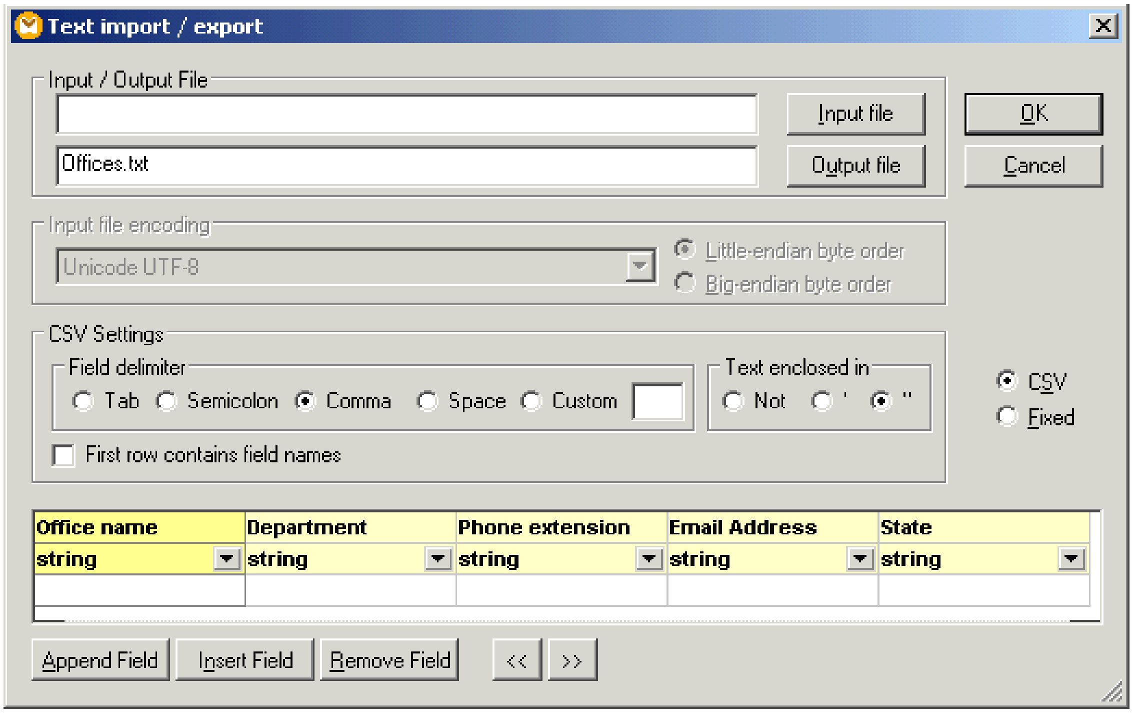
Next / Previous
Clicking one of these buttons moves the currently active column left or right in the preview window.

8.6 Mapping Database to CSV/Text files

This example maps a simple MS Access database, altova.mdb, to a CSV file. The altova.mdb file is available in the **...\MapForceExamples\Tutorial** folder.



The Offices.txt file entry, entered in the Output file field, is the name that is automatically supplied when you click the "Save generated output" icon from the Output tab.



Text import / export

Input / Output File

Output file: Offices.txt

Input file encoding: Unicode UTF-8 ● Little-endian byte order ○ Big-endian byte order

CSV Settings

Field delimiter: ○ Tab ○ Semicolon ● Comma ○ Space ○ Custom

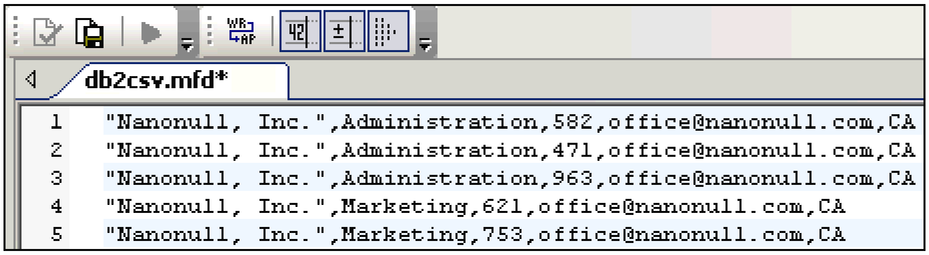
Text enclosed in: ○ Not ○ ' ● "

□ First row contains field names

● CSV ○ Fixed

Office name	Department	Phone extension	Email Address	State
string	string	string	string	string

Click the "Save generated output" icon to generate/output the text file.



db2csv.mfd*

```
1  "Nanonull, Inc.",Administration,582,office@nanonull.com,CA
2  "Nanonull, Inc.",Administration,471,office@nanonull.com,CA
3  "Nanonull, Inc.",Administration,963,office@nanonull.com,CA
4  "Nanonull, Inc.",Marketing,621,office@nanonull.com,CA
5  "Nanonull, Inc.",Marketing,753,office@nanonull.com,CA
```

Chapter 9

Generating XQuery 1.0 code

9 Generating XQuery 1.0 code

MapForce generates XQuery 1.0 program code which can be executed using Altova's XQuery engine in the AltovaXML package, or opened directly in XMLSpy and executed using the menu option **XSL/XQuery | XQuery Execution**.

Please note that execution speed of generated XQuery 1.0 code is significantly faster that of generated XSLT 1.0 / 2.0 code. Generated program code such as Java, C#, or C++, is of course even quicker, because it is compiled before execution.

To download the AltovaXML package (which contains the Altova XQuery engine):

- Point your Browser to http://www.altova.com/download_components.html then select and install the AltovaXQuery engine.

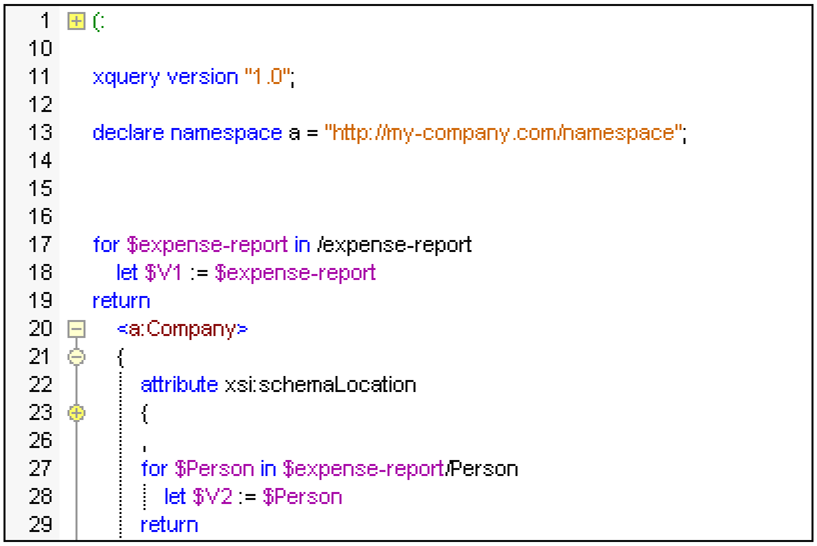
This example uses the **Tut-ExpReport.mfd** file supplied in the ...\MapForceExamples\Tutorial folder. Make sure that you have selected the XQuery XQ icon before you preview the results.

To preview an XQuery result:

Before generating program code it is a good idea to preview the result of the XQuery using the MapForce Engine.

Having opened the Tut-ExpReport.mfd file in MapForce:

1. Click the **XQuery** tab to preview the generated XQuery code.
2. Click the **Output** tab to preview the result of the mapping.



```
(:

xquery version "1.0";

declare namespace a = "http://my-company.com/namespace";



for $expense-report in /expense-report
  let $V1 := $expense-report
return
  <a:Company>
  {
    attribute xsi:schemaLocation
    {
    ,
    for $Person in $expense-report/Person
      let $V2 := $Person
    return
```

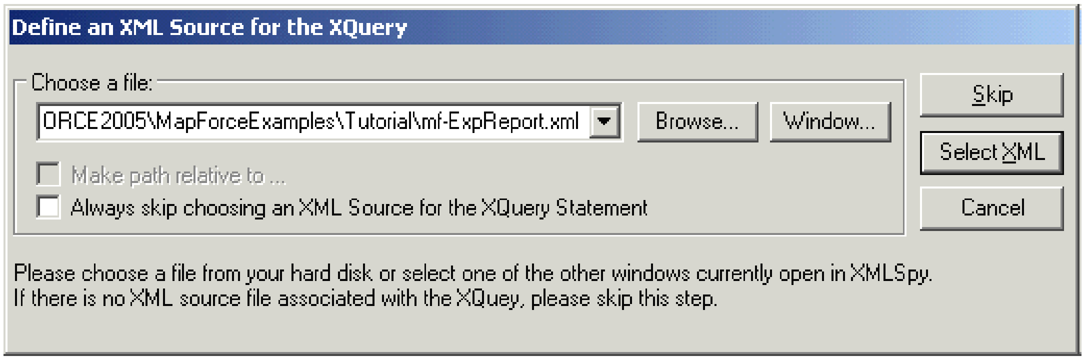
To generate XQuery code:

1. Open the **Tut-ExpReport.mfd file** in MapForce.
2. Select the menu item **File | Generate code in | XQuery**.
3. Select the folder you want to place the generated XQuery file in, (e.g. ...\MapForceExamples) and click OK.
   A message appears showing that the generation was successful.
4. Navigate to the designated folder and you will find the XQuery file with the file name **MapToExpReport-Target.xq**.

To execute the XQuery using XMLSpy:

1. Start XMLSpy and open the previously generated MapToExpReport-Target.xq file.

2. Select the menu option **XSL/XQuery | XQuery execution**.



Define an XML Source for the XQuery

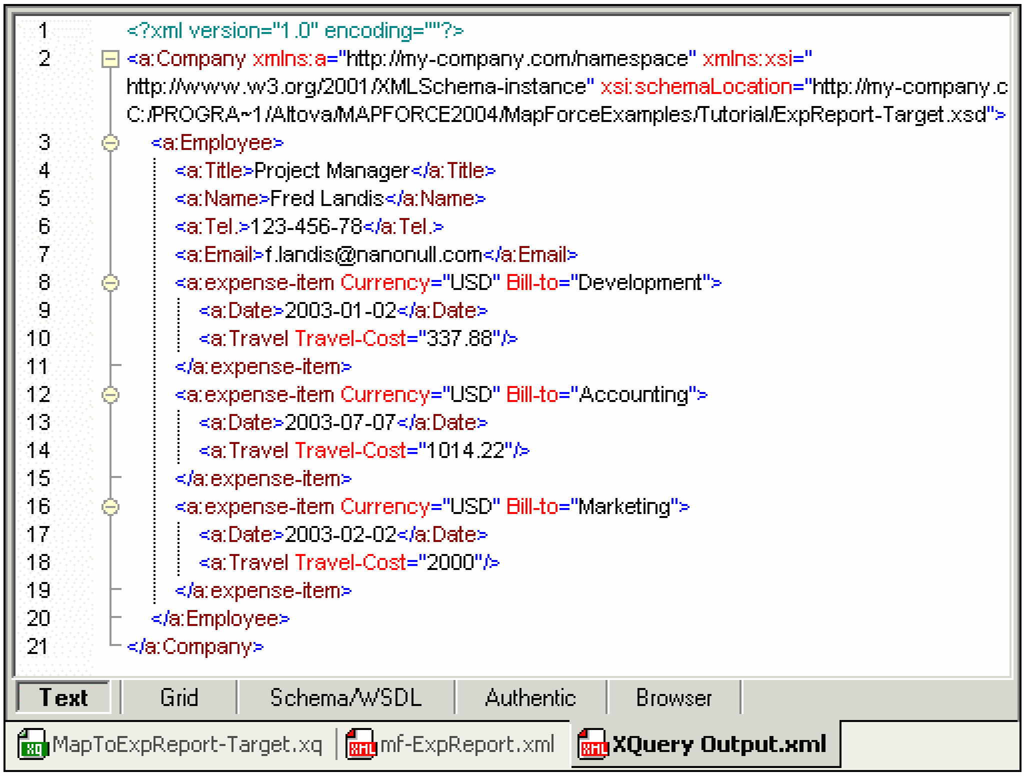
Choose a file: ORCE2005\MapForceExamples\Tutorial\mf-ExpReport.xml

□ Make path relative to ...

□ Always skip choosing an XML Source for the XQuery Statement

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.
If there is no XML source file associated with the XQuey, please skip this step.

3. Click the Browse button and select the XML file that is to act as the data source, e.g. mf-ExpReport.xml.
4. Click the "Select XML" button to execute the XQuery.



```
<?xml version="1.0" encoding=""?>
<a:Company xmlns:a="http://my-company.com/namespace" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://my-company.c
C:/PROGRA~1/Altova/MAPFORCE2004/MapForceExamples/Tutorial/ExpReport-Target.xsd">
	<a:Employee>
		<a:Title>Project Manager</a:Title>
		<a:Name>Fred Landis</a:Name>
		<a:Tel.>123-456-78</a:Tel.>
		<a:Email>f.landis@nanonull.com</a:Email>
		<a:expense-item Currency="USD" Bill-to="Development">
			<a:Date>2003-01-02</a:Date>
			<a:Travel Travel-Cost="337.88"/>
		</a:expense-item>
		<a:expense-item Currency="USD" Bill-to="Accounting">
			<a:Date>2003-07-07</a:Date>
			<a:Travel Travel-Cost="1014.22"/>
		</a:expense-item>
		<a:expense-item Currency="USD" Bill-to="Marketing">
			<a:Date>2003-02-02</a:Date>
			<a:Travel Travel-Cost="2000"/>
		</a:expense-item>
	</a:Employee>
</a:Company>
```

An "XQuery Output.xml" file is created which contains the mapped data. Please see the **Altova XQuery Engine** documentation for more information on the command line parameters.

Chapter 10

User-defined functions

10 User-defined functions

MapForce allows you to create user-defined functions (within user-defined libraries) which can contain any number of input and outputs where any of these can be in the form of: simple values, XML nodes, or databases.

There are two types of user-defined functions: those defined as "Inline" and the others as "Standard", please see Inline vs. Standard user-defined functions for more information. Also note that user-defined functions can be changed from the one type to the other.

The main use of user-defined functions is to combine data sources, as well as input and output components, into a single user-defined function / component, which can be used across different mappings.

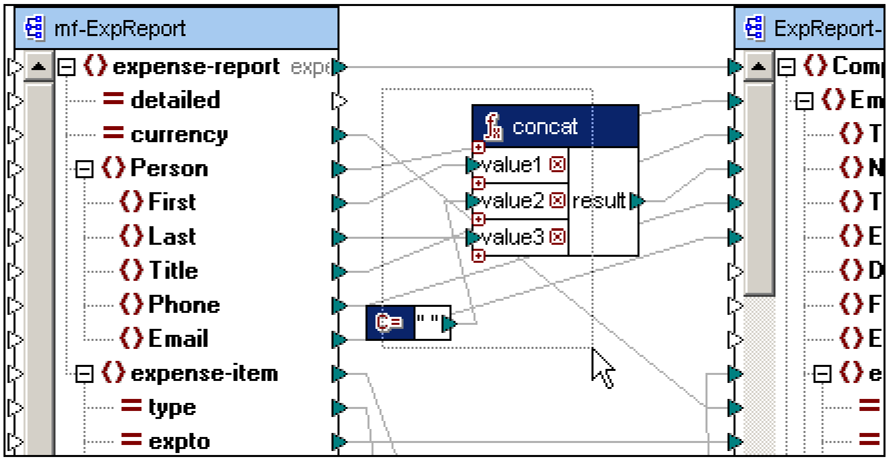
User-defined functions can be:

- built from scratch, or
- use functions currently available in the mapping tab.

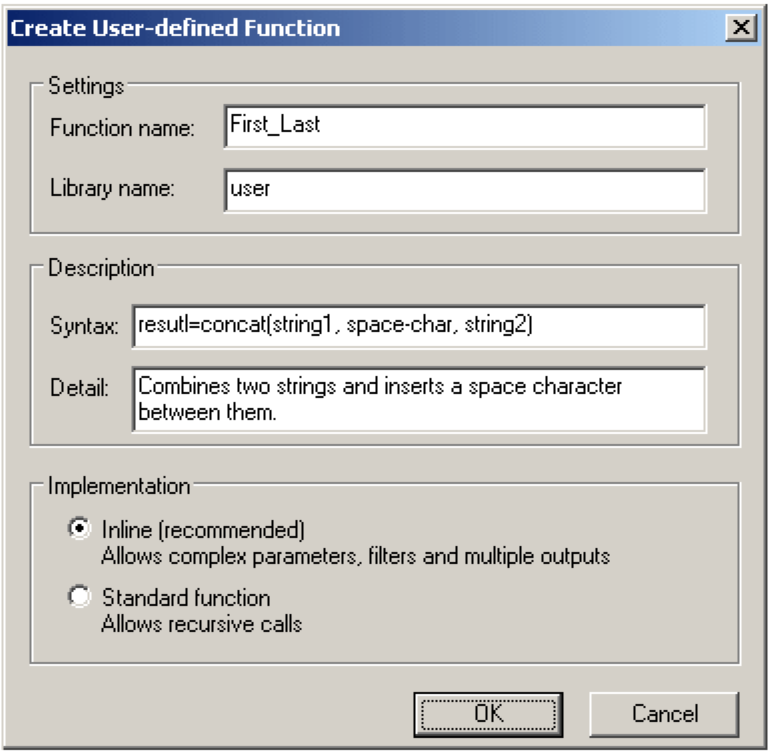
This example uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples folder.

To create a user defined function:

1. Drag to mark both the concat and the constant functions (you can also hold down the CTRL key and click the functions individually).



2. Select the menu option **Function | Create User-Defined Function from Selection**.
3. Enter the name of the new user-defined function (First_Last).
   Note: valid characters are: alphanumeric, a-z, A-Z, 0-9 as well as underscore "_", hyphen/dash "-" and colon ":".
4. Use the Syntax and Detail fields to add extra information on the new function, and click OK to confirm.
   The library name "user" is supplied as a default, you can of course define your own library name in this field.



Create User-defined Function

Settings

Function name: First_Last

Library name: user

Description

Syntax: resutl=concat(string1, space-char, string2)

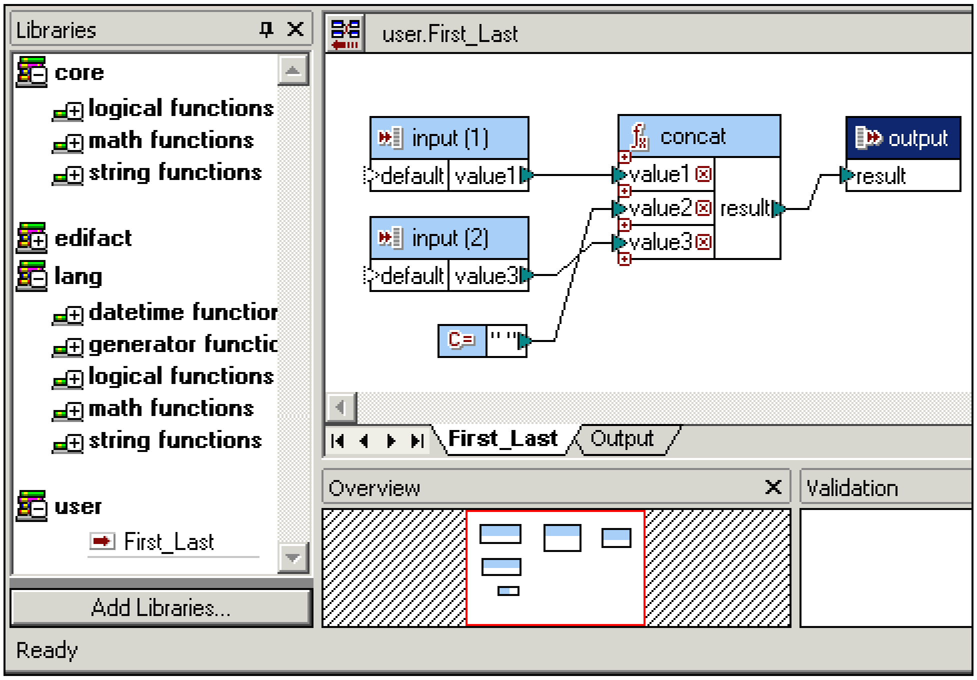
Detail: Combines two strings and inserts a space character between them.

Implementation

● Inline (recommended)
Allows complex parameters, filters and multiple outputs

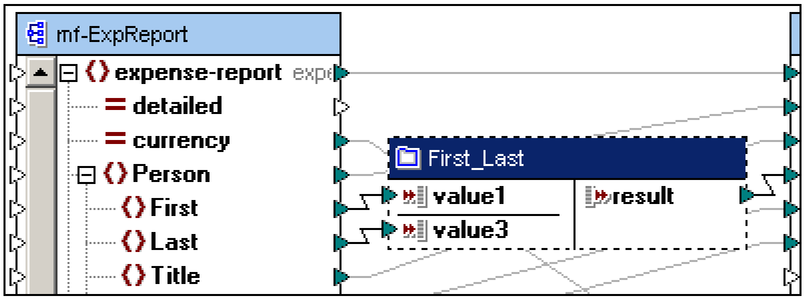
○ Standard function
Allows recursive calls

The individual elements that make up the function group appear in a tab with the function name. The new library "user" appears in the Libraries pane with the function name "First_Last" below it.



Click the Home button to return to the mapping window. The three functions have now been combined into a single function called First_Last.

User-defined functions of type "Inline" are displayed with a dashed outline.



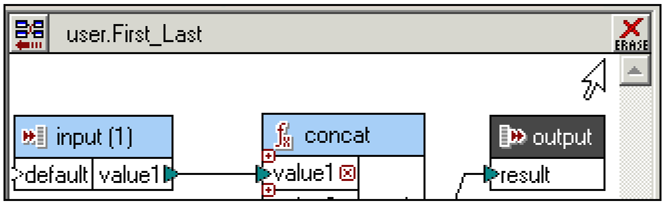
Connect the First and Last items to the input parameters of the user-defined function, and the result parameter to the Name item. Dragging on the function name in the Libraries pane and dropping it in the mapping window, allows you to use it elsewhere.

Please note:

Double clicking a user-defined function, displays the individual components in a tab of that name. User-defined functions can be defined to contain complex inputs/outputs (XML nodes etc.) as well as multiple output components. Please see "Standard user-defined function" and "Complex user-defined function" for more information.

To delete a user-defined function from a library:

1. Double click the specific user-defined function in the Libraries window. The user-defined function is visible in its tab.
2. Click the **Erase** button in the title bar to delete the function.



Reusing - exporting and importing User-defined functions:

User-defined functions, defined in one mapping, can be imported into any other mapping:

1. Click the **Add Libraries** button and select a previously defined ***.mfd** file, that contains the user-defined function(s) you want to import. The user-defined functions now appear in the Libraries window (under "user" if that is the default library you selected).
2. Drag the imported function into the mapping to make use of it.

To change the user-defined function "type":

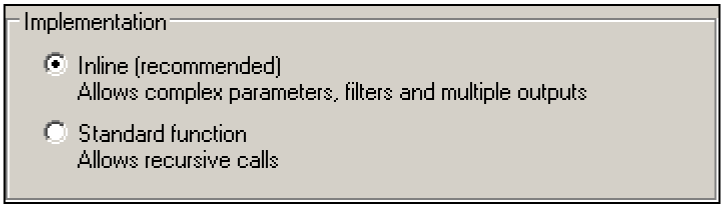
1. Double click the user-defined function to see its constituent components.
2. Select the menu option **Function | Function settings** and click the radio button of the type you want to change it to, Standard or Inline.

10.1 Inline vs. Standard user-defined functions

The main difference between these two types of functions is the level of complexity that they each support, and the implementation of each during code generation.

The graphical representation of the two types also differ:

- Standard user-defined functions are shown with a **solid** outline
- Inline user-defined functions are shown with a **dashed** outline



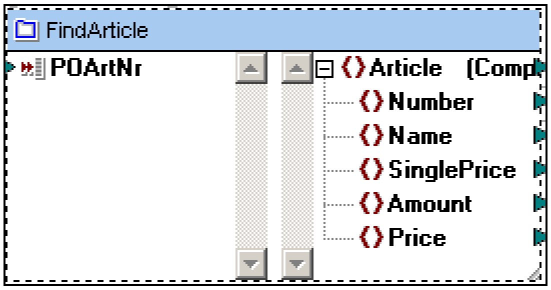
Implementation

● Inline (recommended)
Allows complex parameters, filters and multiple outputs

○ Standard function
Allows recursive calls

Inline user-defined functions **support**:

- Complex input and output components i.e. XML schema nodes, databases etc.
- Multiple output components within a function
- Direct connection of filters to input components
- Exist type functions on input component e.g. exists, not exists, substitute-missing is-null, is not null, substitute-null.



Inline user-defined functions **do not support**:

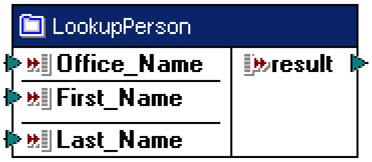
- The setting of a priority context on a parameter
- Recursive calls to user-defined functions

Code generation:
In essence an inline user-defined component implements the constituent components of the user-defined function instead of generating a function call. All parameters are evaluated and purged.

If the user-defined function is defined as inlined, filters and exists-like functions can be used because MapForce generates code that works exactly as the function's constituent components.

Standard user-defined function **support**:

- Only simple input components
- Only a single output component
- Recursive calls to user-defined functions (where the exit condition must be supplied, e. g. use an If-Else condition where one branch, or value, exits the recursion)
- Setting a priority context on a parameter



Please note:
Although Standard user-defined functions **do not** support complex input and output components, they **can be created** in this type of function. An error message appears when you try to preview the result of the mapping, and prompts if you want to change the current Standard type user-defined function, into one of type "Inline".

Standard user-defined functions **do not support**:

- Complex input and output components i.e. XML schema nodes, databases etc.
- Direct connection of filters to input components
- Exist type functions on input components:
  - Exists
  - Not exists
  - Substitute-missing
  - is-null, is not null, substitute-null

  Code generation:
  A standard user-defined component generates code for a function call, where inputs and outputs are passed as parameters. At runtime, the input parameter values are evaluated first, then the function is called for each occurrence of the input data.

To change the user-defined function "type":

1. Double click the user-defined function to see its constituent components.
2. Select the menu option **Function | Function settings** and click the radio button of the type you want to change it to, Standard or Inline.

Please note:
If the user-defined function was originally of type "standard" with a priority context, and was subsequently changed to one of type "inline", then the priority context is hidden and deactivated. Changing the same function back to "standard", shows the priority context and enables it once again.

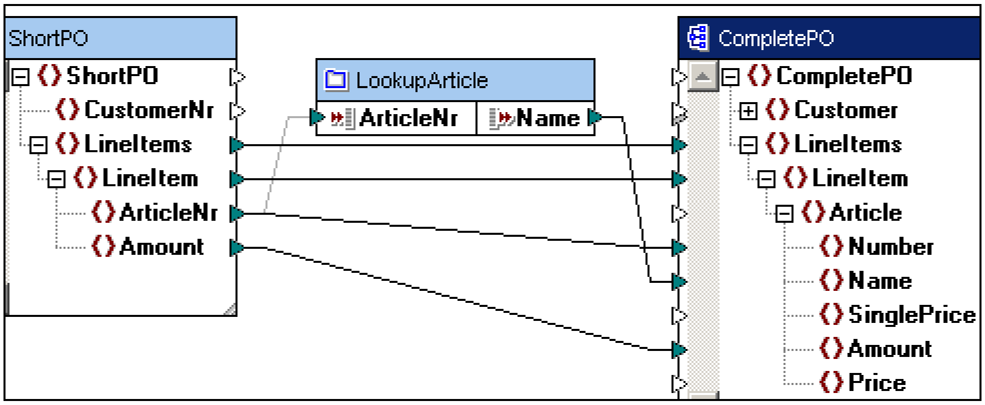
User-defined functions and Copy-all connections
When creating Copy-all connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

10.2 Standard user-defined function

This example is provided as the **lookup-standard.mfd** file available in the ...\ **MapForceExamples** folder.

Aim:
To create a generic look-up function that:

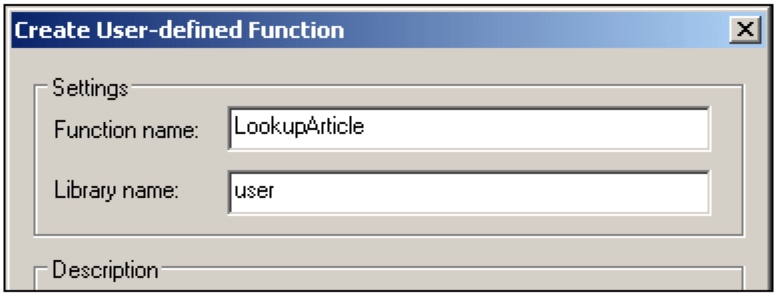
- supplies Articles/Number data from the Articles XML file, to be compared to Article numbers of a different XML file, ShortPO in this case.



- Insert the ShortPO.xsd and assign ShortPO.xml as the source XML file.
- Insert the CompletePO.xsd schema file, and select CompletePO as the root element.
- Insert a new user-defined function using the method described below.

To create a user defined function from scratch:

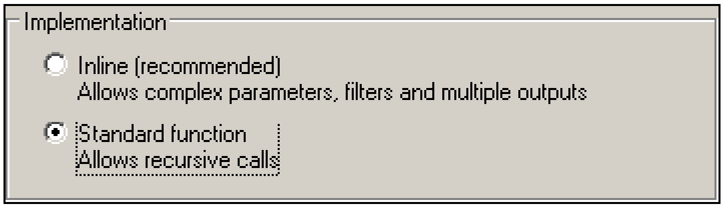
1. Select the menu option **Function | Create User-defined function**.
2. Enter the name of the function e.g. LookupArticle.


Create User-defined Function

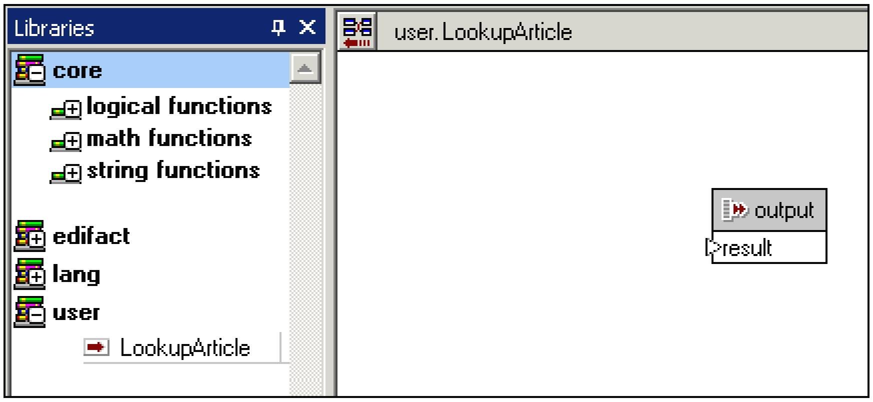
Settings
Function name: LookupArticle
Library name: user

Description

3. Select the "Standard function" radio button and click OK to confirm


Implementation
○ Inline (recommended)
Allows complex parameters, filters and multiple outputs
● Standard function
Allows recursive calls

A tab only containing only one item, an output function, is displayed.

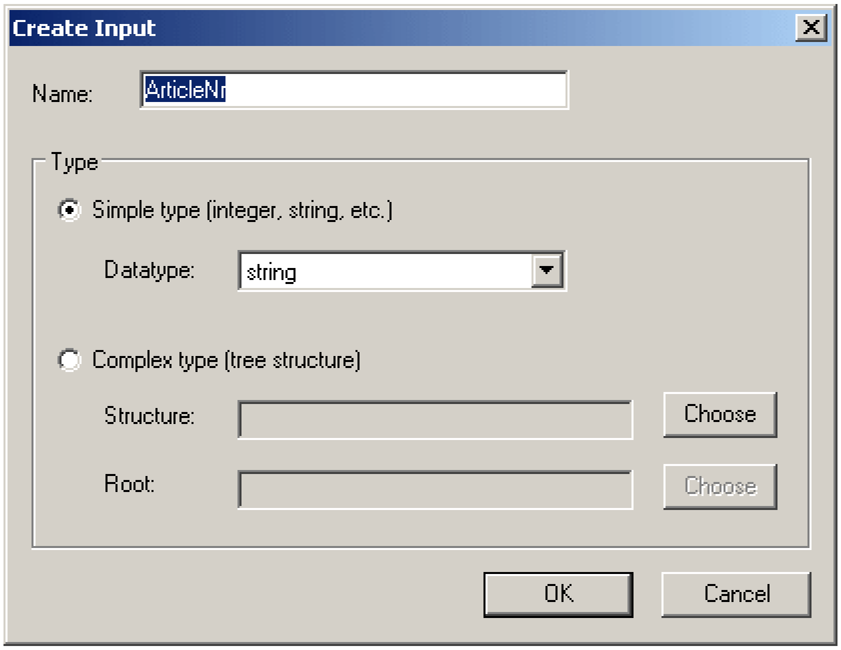


This is the working area used to define the user-defined function.

A new library has been created in the Libraries pane with the name "user" and the function name "LookupArticle".

3. Click the **Insert Schema/XML file** icon to insert the **Articles** schema and select the XML file of the same name to act as the data source.

4. Click the **Insert input component** icon to insert an input component.
5. Enter the name of the input parameter, ArticleNr in this case, and click OK.



Name: ArticleNr

● Simple type (integer, string, etc.)

Datatype: string

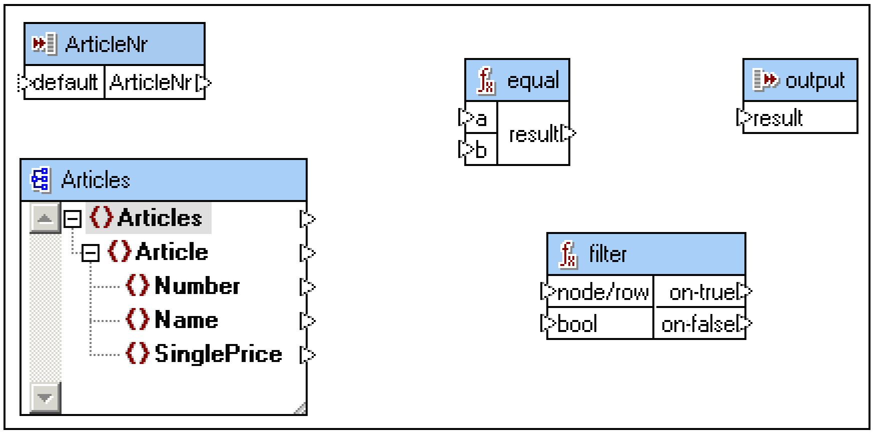
○ Complex type (tree structure)

Structure:

Root:

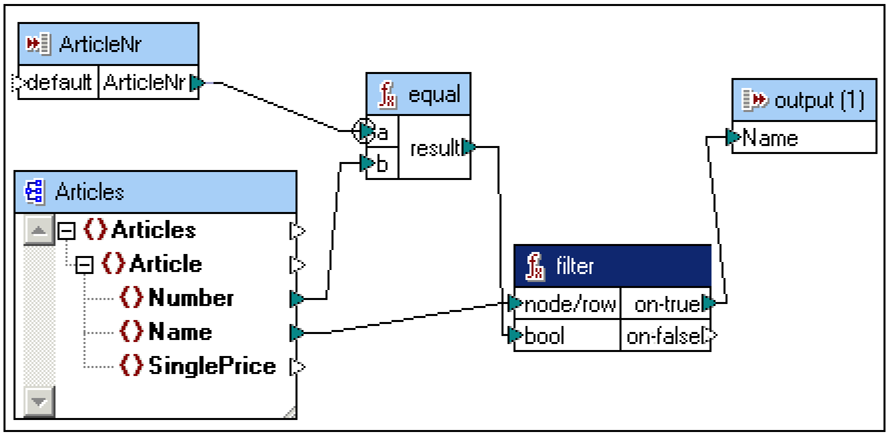
This component acts as a data input to the user-defined function and supplies the input icon of the user-defined function.
6. Insert an "**equal**" component by dragging it from the core library/logical functions group.

7. Insert a **filter** component by clicking the Insert Filter icon in the toolbar.



Use the diagram below as an aid to creating the mappings in the user-defined function, please take note of the following:

8 Right click the **a** parameter and select **Priority context** from the pop up menu.
9. **Double click** the output function and enter the name of the output parameter, in this case "**Name**".



This ends the definition of the user-defined function.

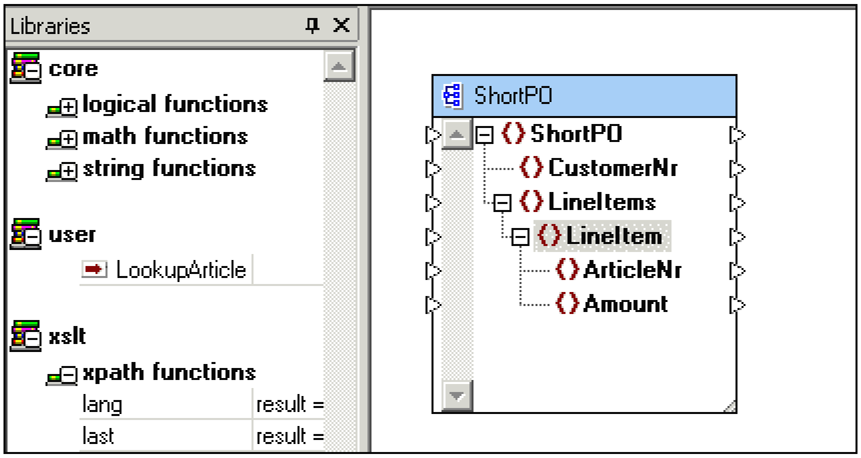
Please note:

Double clicking the input and output functions opens a dialog box in which you can change the datatype of the input parameter, as well as define if the function is to have an input icon (Connection required) in this dialog.

The user-defined function:

- has one **input** function, ArticleNr, which receives data from the ShortPO XML file.
- **compares** the ShortPO ArticleNr, with the Article/Number from the **Articles** input XML instance file, inserted into the user-defined function for this purpose.
- uses a **filter** component to forward the Article/Name records to the output component, if the comparison returns true.
- has one output function, Name, which forwards the Article Name records to the CompletePO XML file.

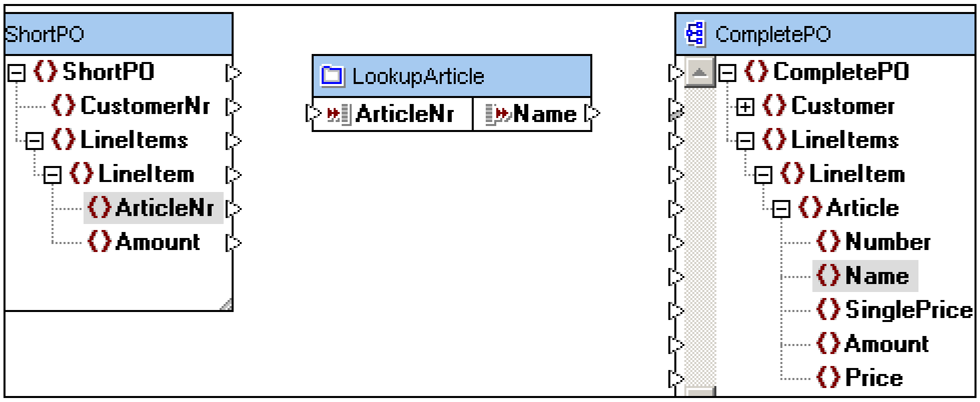
10. Click the Home icon to return to the mapping.
The LookupArticle user-defined function, is now available under the user library.



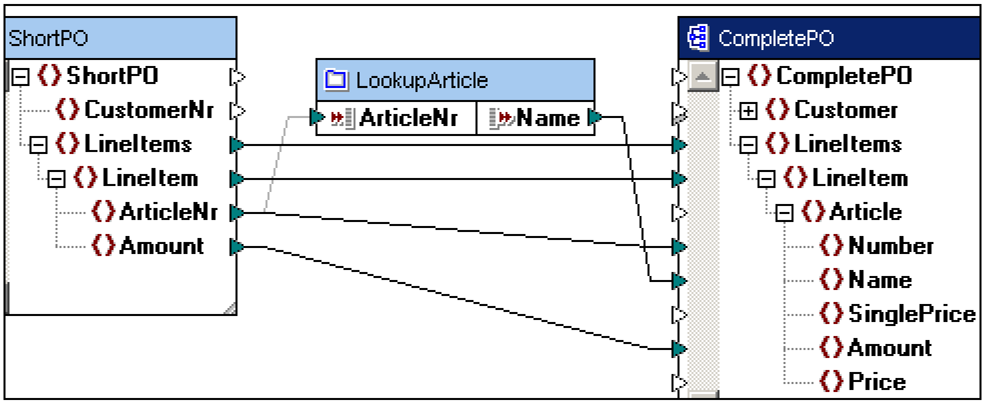
11. Drag the LookupArticle function into the Mapping window.

The user-defined function is displayed:

- with its name "LookupArticle" in the title/function bar,
- with named input and output icons.



10. Create the mappings displayed in the graphic below and click the Output tab to see the result of the mapping.



Please note:

Using **filters** in user-defined functions only make sense if the source-component is also in the same user-defined function.

Filters can only be used to supply data **into** a user-defined function using input components, if you have defined it as an inline function.

10.3 Complex user-defined function - XML node as input

This example is provided as the **lookup-udf-in.mfd** file available in the ...**MapForceExamples** folder. What this section will show, is how to define an inline user-defined function that contains a complex input components.

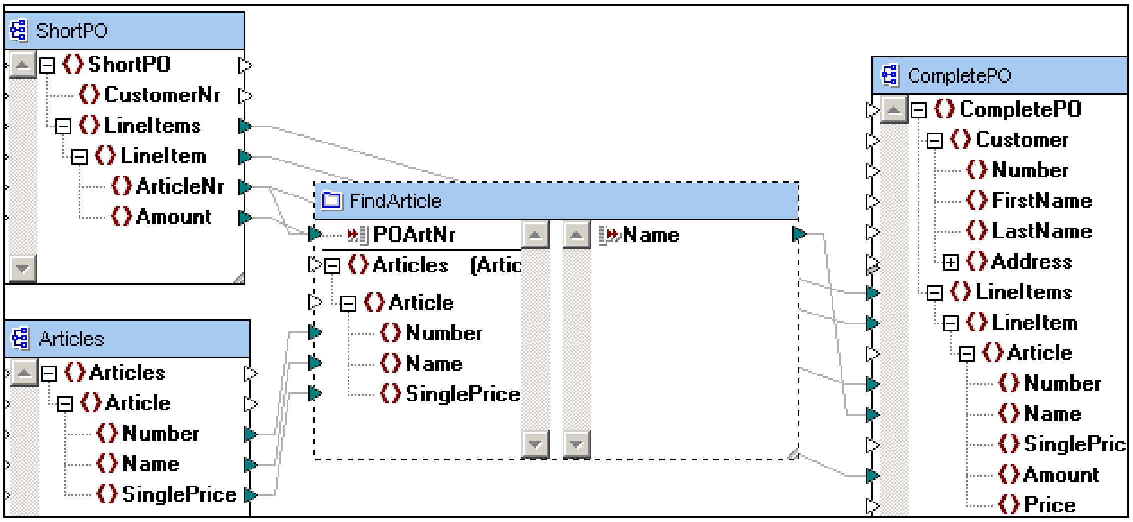
Note that the user-defined function "FindArticle" consists of two halves.

A left half which contains the input parameters:

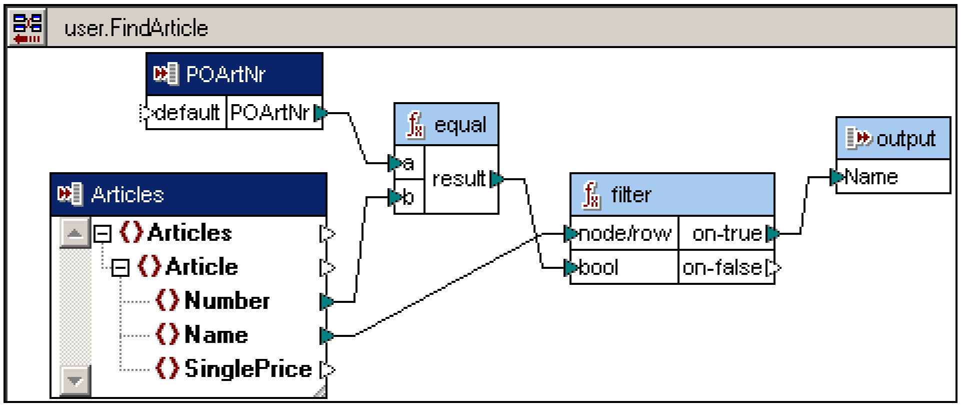
- a simple input parameter **POArtNr**
- a complex input component **Articles,** with mappings directly to its XML child nodes

A right half which contains:

- a simple output parameter called "**Name"**.



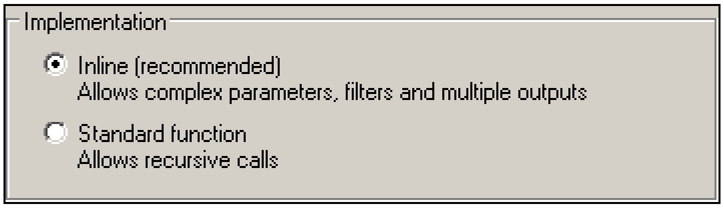
The screenshot below shows the constituent components of the user-defined function, the two input components at left and the output component at right.



10.3.1 Complex input components - defining

Defining complex input components:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click Enter to confirm. Note that the **Inline...** option is automatically selected.

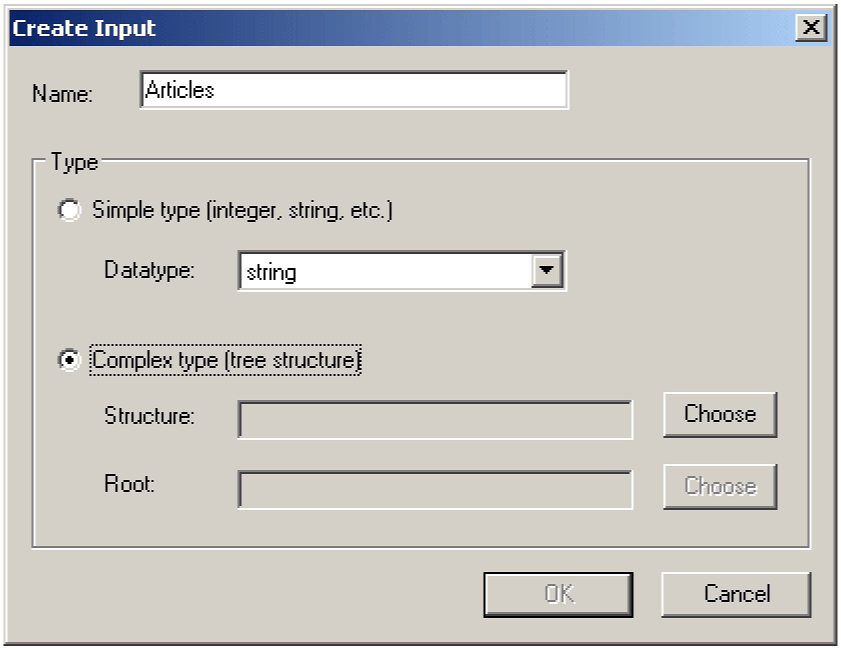


Implementation

● Inline (recommended)
Allows complex parameters, filters and multiple outputs

○ Standard function
Allows recursive calls

2. Click the **Insert input component** icon in the icon bar.
3. Enter the name of the component into the Name field.



Create Input

Name: Articles

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

Structure:

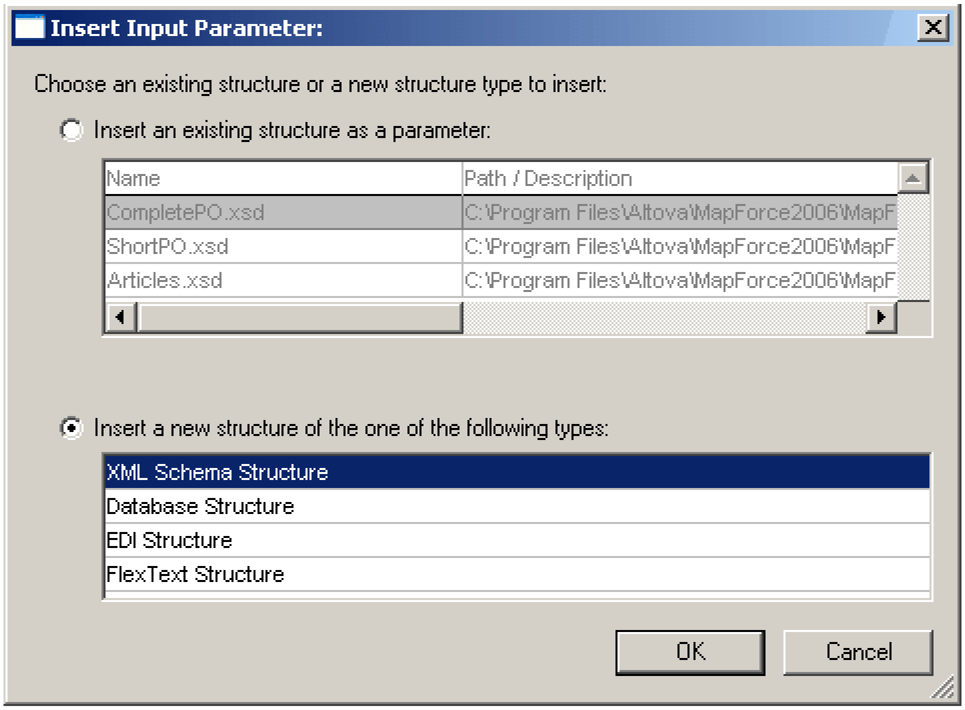
Root:

4. Click the **Complex type** radio button, then click the "Choose" button next to the Structure field.
   This opens the "Insert Input Parameter" dialog box.

   The top list box displays the **existing** components in the mapping, in this example three schemas. Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML schema, database file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, file.

5. Click "Insert new structure... " radio button, select the XML Schema structure entry, and click OK to continue.



Insert Input Parameter:

Choose an existing structure or a new structure type to insert:

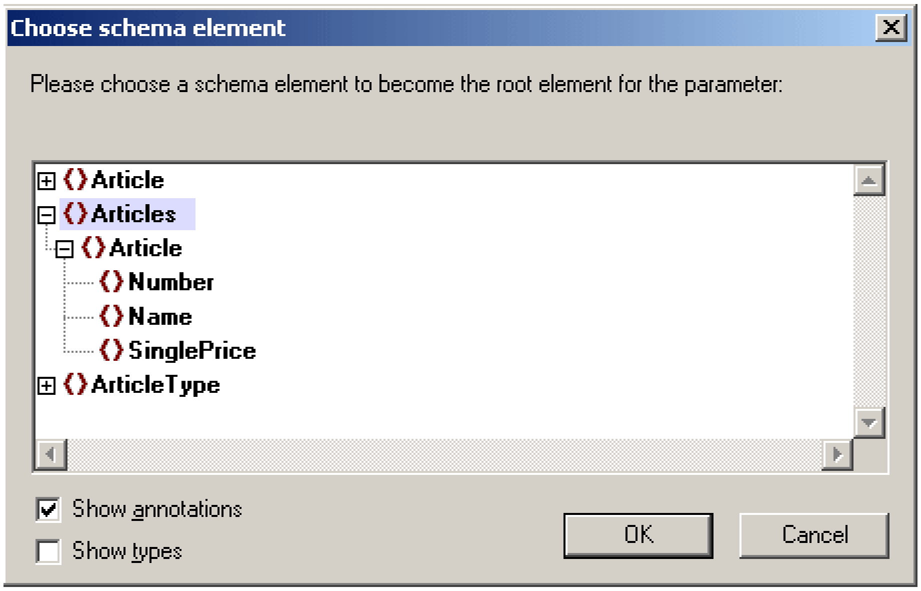
○ Insert an existing structure as a parameter:

Name	Path / Description
CompletePO.xsd	C:\Program Files\Altova\MapForce2006\MapF
ShortPO.xsd	C:\Program Files\Altova\MapForce2006\MapF
Articles.xsd	C:\Program Files\Altova\MapForce2006\MapF

● Insert a new structure of the one of the following types:

- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure

6. Select the Articles.xsd from the "Open" dialog box.
7. Click the element that you would like to become the root element in the component, e.g. Articles, and click OK to confirm.



Choose schema element

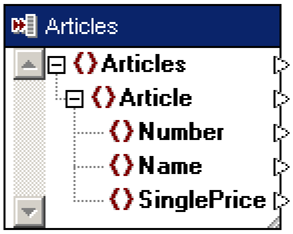
Please choose a schema element to become the root element for the parameter:

```
⊞ {} Article
⊟ {} Articles
  ⊟ {} Article
      {} Number
      {} Name
      {} SinglePrice
⊞ {} ArticleType
```

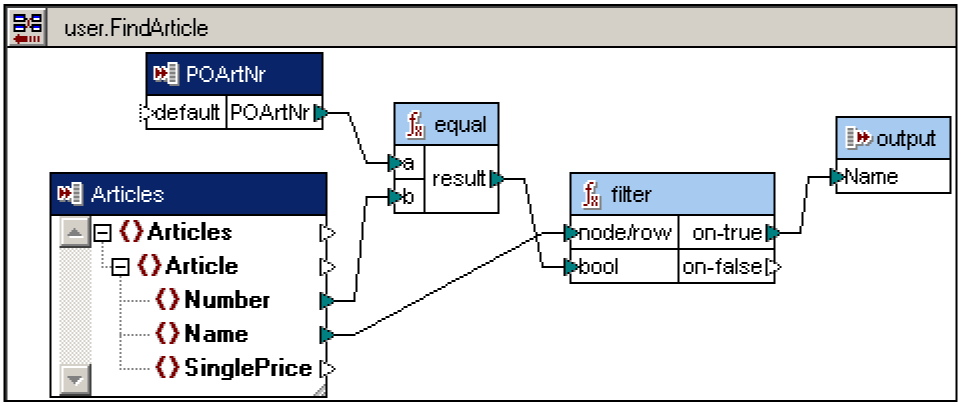
☑ Show annotations

☐ Show types

The Articles component is inserted into the user-defined function. Please note the input icon to the left of the component name. This shows that the component is used as a complex input component.



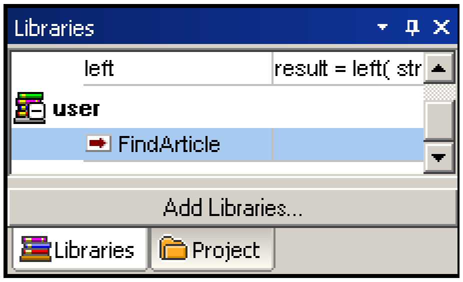
8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component, filter, equal and output components, and connect them as shown.



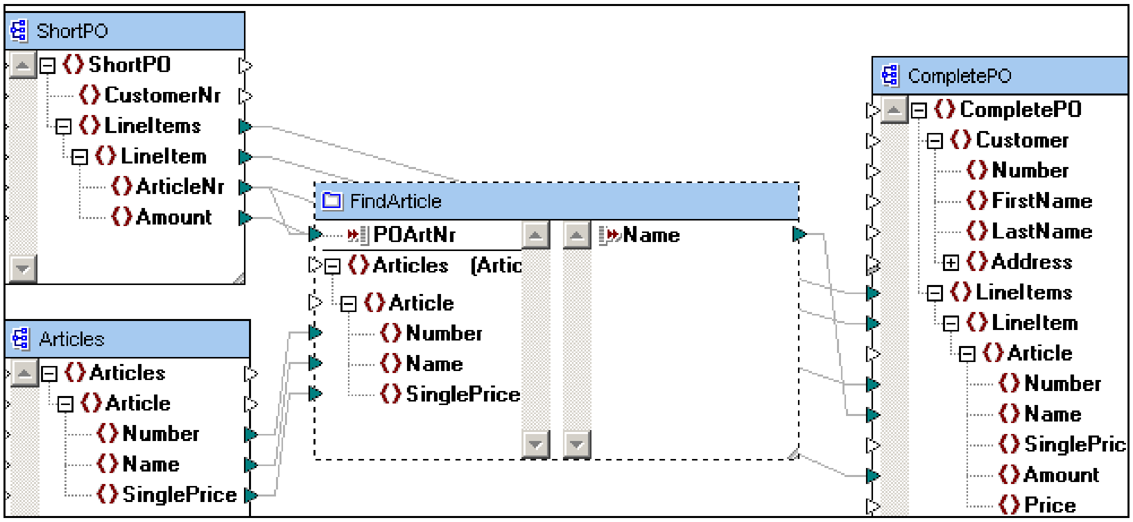
Please note:

- The Articles input component receives its data from outside of the user-defined function. Input icons that allow mapping to this component, are available there.
- An XML instance file to provide data from within the user-defined function, cannot be assigned to a complex input component.
- The other input component input(1), supplies the ShortPO article number data to which the Articles | Number is compared.
- The filter component filters out the records where both numbers are identical, and passes them on to the output component.

10. Click the Home icon to return to the mapping.
11. Drag the newly created user-defined component from the Libraries pane into the mapping.



12. Create the connections as shown in the screenshot below.

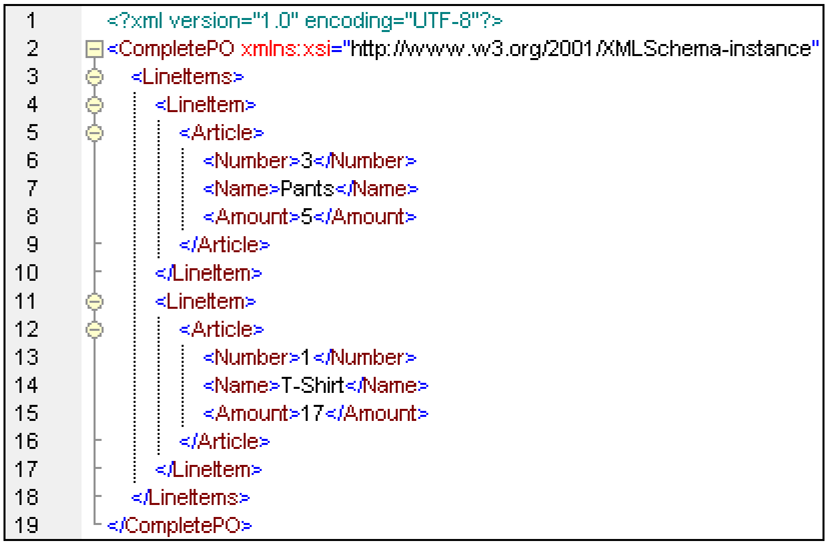


The left half contains the input parameters to which items from two schema/xml files are mapped:

- ShortPO supplies the data for the input component **POArtNr**
- Articles supplies the data for the complex input component. The Articles.xml instance file was assigned to the Articles schema file when the component was inserted.
- The complex input component **Articles** with its XML child nodes, to which data has been mapped from the Articles component.

The right half contains:

- a simple output parameter called "**Name"**, which passes on the filtered line items which have the same Article number, to the Name item of Complete PO.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <LineItems>
        <LineItem>
            <Article>
                <Number>3</Number>
                <Name>Pants</Name>
                <Amount>5</Amount>
            </Article>
        </LineItem>
        <LineItem>
            <Article>
                <Number>1</Number>
                <Name>T-Shirt</Name>
                <Amount>17</Amount>
            </Article>
        </LineItem>
    </LineItems>
</CompletePO>
```

Please note:
When creating **Copy-all** connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however.

10.4 Complex user-defined function - XML node as output

This example is provided as the **lookup-udf-out.mfd** file available in the ...\ **MapForceExamples** folder. What this section will show is how to define an inline user-defined function that allows a complex output component.

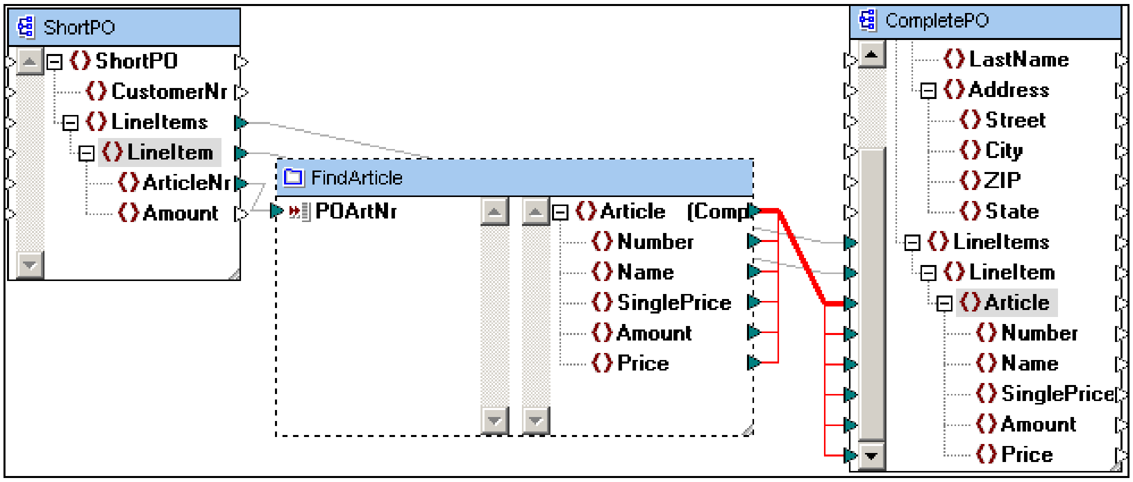
Note that the user-defined function FindArticle consists of two halves.

A left half which contains the input parameter:

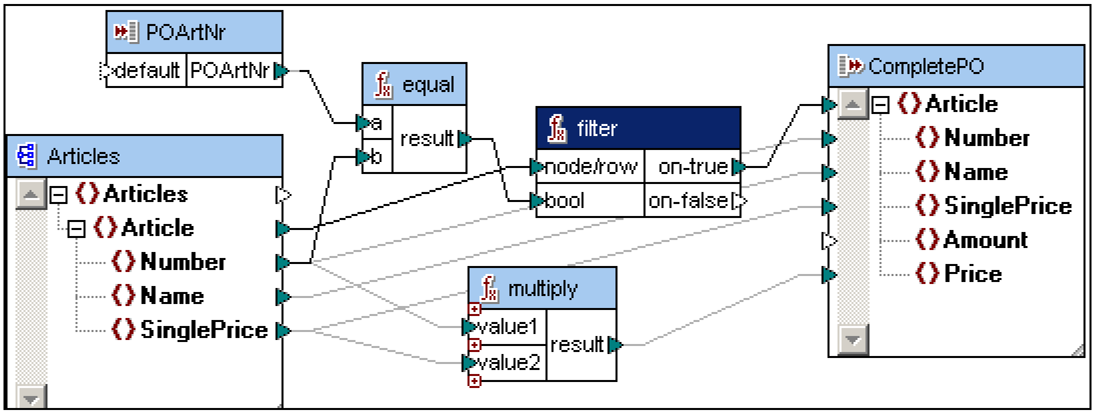
- a simple input parameter **POArtNr**

A right half which contains:

- a complex output component **Article (CompletePO)** with its XML child nodes mapped to CompletePO.



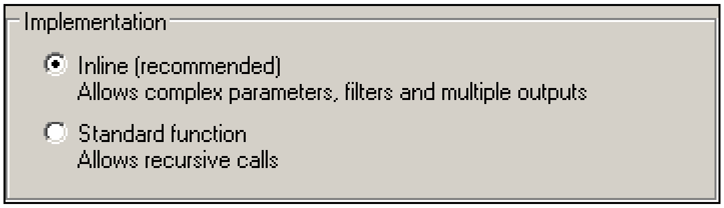
The screenshot below shows the constituent components of the user-defined function, the input component at left and the complex output component at right.



10.4.1 Complex output components - defining

Defining complex output components:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click Enter to confirm. Note that the **Inline...** option is automatically selected.

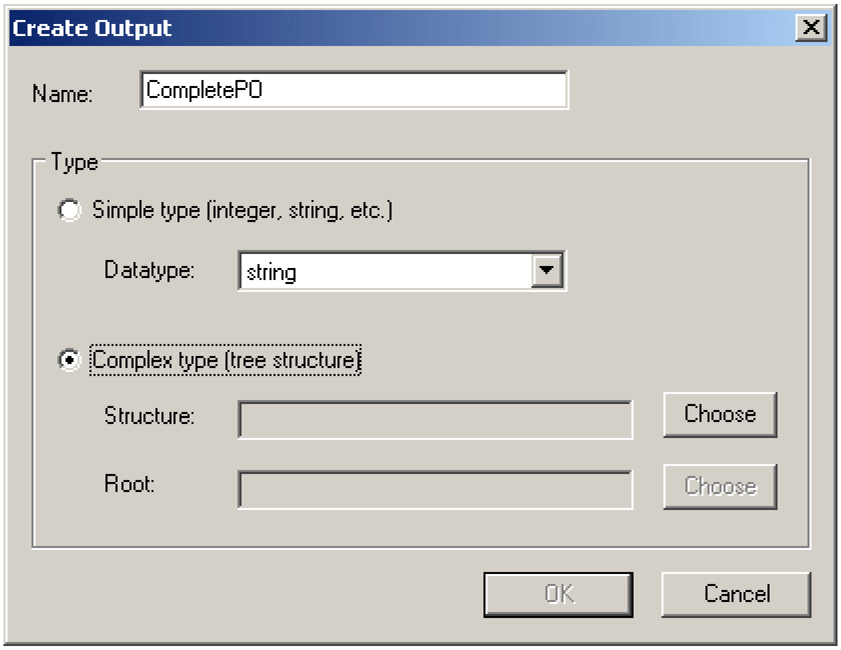


Implementation

● Inline (recommended)
Allows complex parameters, filters and multiple outputs

○ Standard function
Allows recursive calls

2. Click the Insert output icon in the icon bar, and enter a name e.g. CompletePO.



Name: CompletePO

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

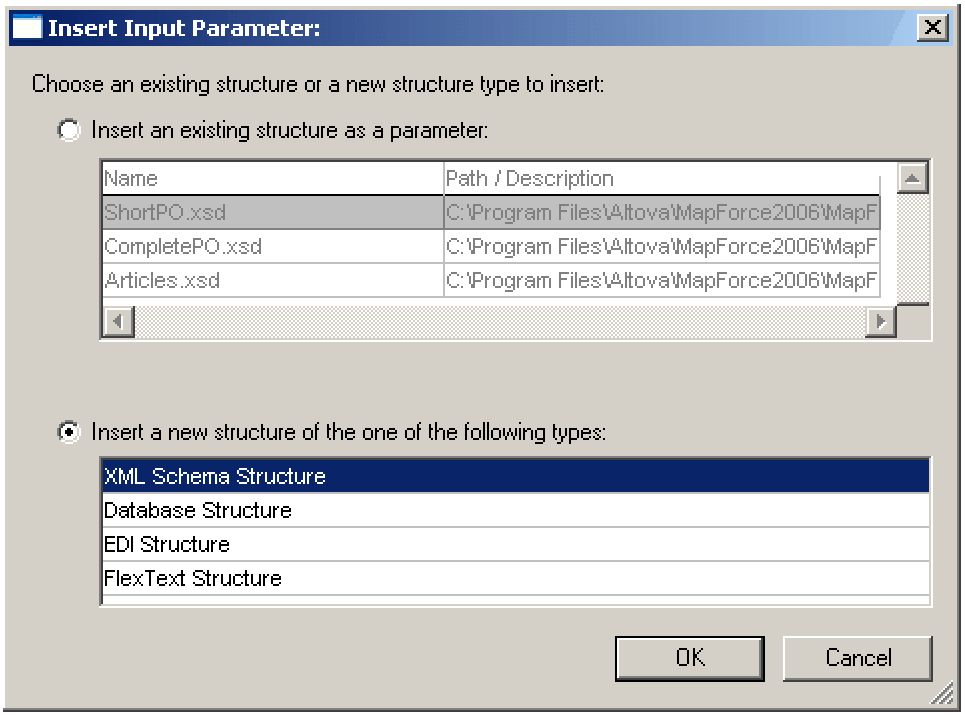
Structure:

Root:

3. Click the **Complex type** radio button, then click the "Choose" button.
This opens the "Insert Input Parameter" dialog box.

   The top list box displays the **existing** components in the mapping, in this example three schemas. Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML Schema , database file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema, Database file, file.



Insert Input Parameter:

Choose an existing structure or a new structure type to insert:

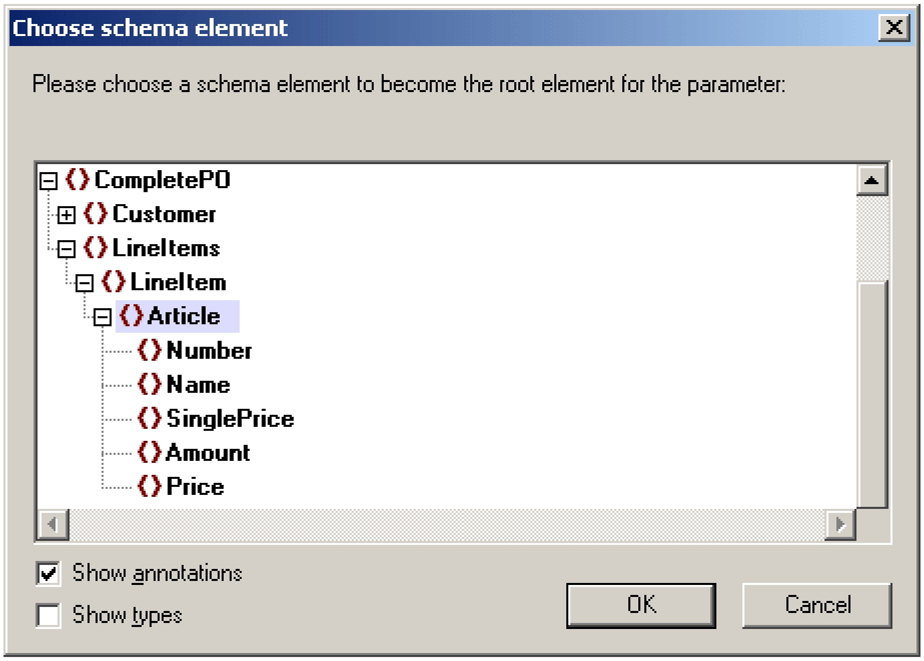
○ Insert an existing structure as a parameter:

Name	Path / Description
ShortPO.xsd	C:\Program Files\Altova\MapForce2006\MapF
CompletePO.xsd	C:\Program Files\Altova\MapForce2006\MapF
Articles.xsd	C:\Program Files\Altova\MapForce2006\MapF

● Insert a new structure of the one of the following types:

- XML Schema Structure
- Database Structure
- EDI Structure
- FlexText Structure

4. Click "Insert new structure... " radio button, select the XML Schema structure entry, and click OK to continue.
5. Select the CompletePO.xsd from the "Open" dialog box.
6. Click the element that you would like to become the root element in the component, e.g. Article, and click OK to confirm.



Choose schema element

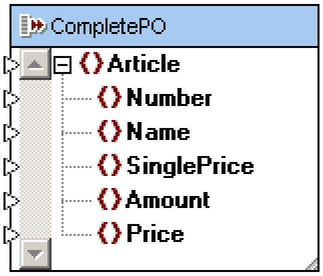
Please choose a schema element to become the root element for the parameter:

- CompletePO
  - Customer
  - LineItems
    - LineItem
      - Article
        - Number
        - Name
        - SinglePrice
        - Amount
        - Price

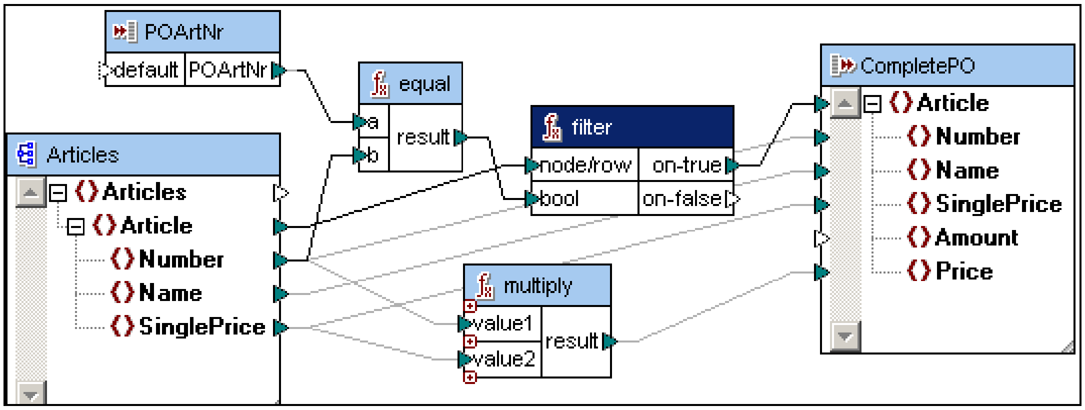
☑ Show annotations

□ Show types

The CompletePO component is inserted into the user-defined function. Please note the output icon to the left of the component name. This shows that the component is used as a complex output component.



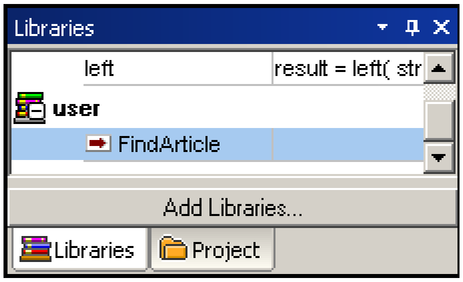
7. Insert the Articles schema/XML file into the user-defined function and assign the Articles.xml as the XML instance.
8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component, filter, equal and multiply components, and connect them as shown.



Please note:

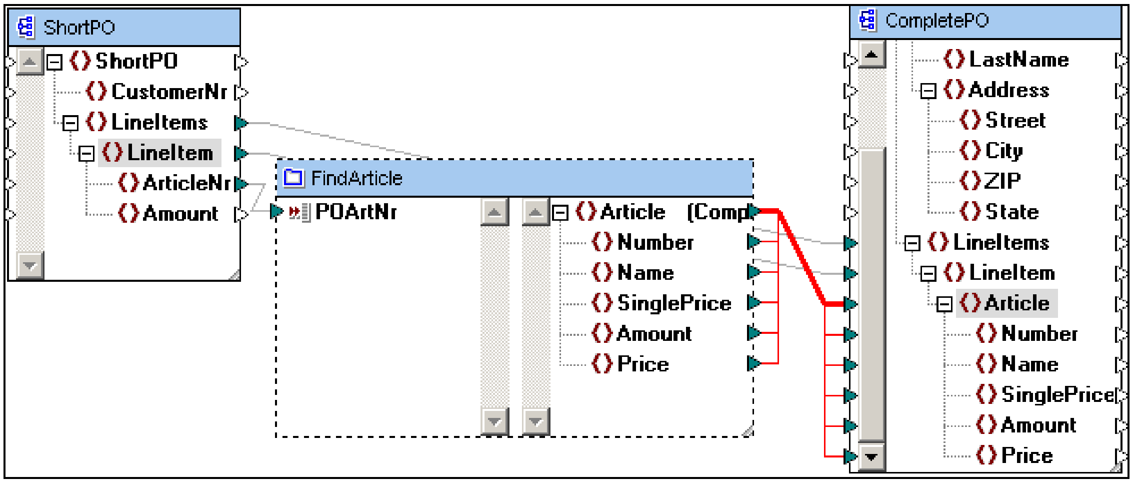
- The Articles component receives its data from the Articles.xml instance file, within the user-defined function.
- The input component input(1), supplies the ShortPO article number data to which the Articles | Number is compared.
- The filter component filters out the records where both numbers are identical, and passes them on to the CompletePO output component.

9. Click the Home icon to return to the mapping.
10. Drag the newly created user-defined component from the Libraries pane into the mapping.



11. Create the connections as shown in the screenshot below.
Having created the Article connector, right click it and select "Copy-all" from the context

menu. The rest of the connectors are automatically generated, and are highlighted in the screenshot below.



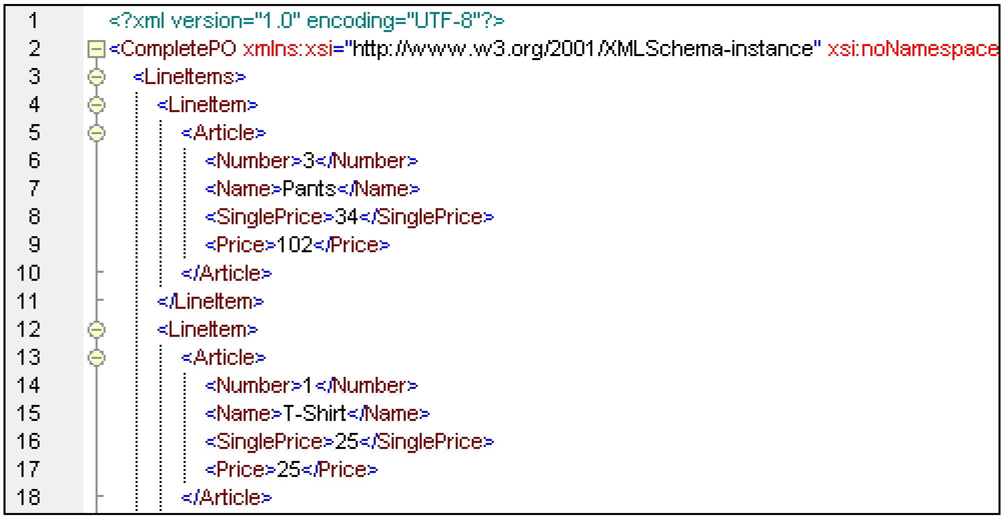
Please note:
When creating Copy-all connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however.

The left half contains the input parameter to which a single item is mapped:

- ShortPO supplies the article number to the **POArtNr** input component.

The right half contains:

- a complex output component called "Article (CompletePO)" with its XML child nodes, which maps the filtered items, of the same Article number, to CompletePO.

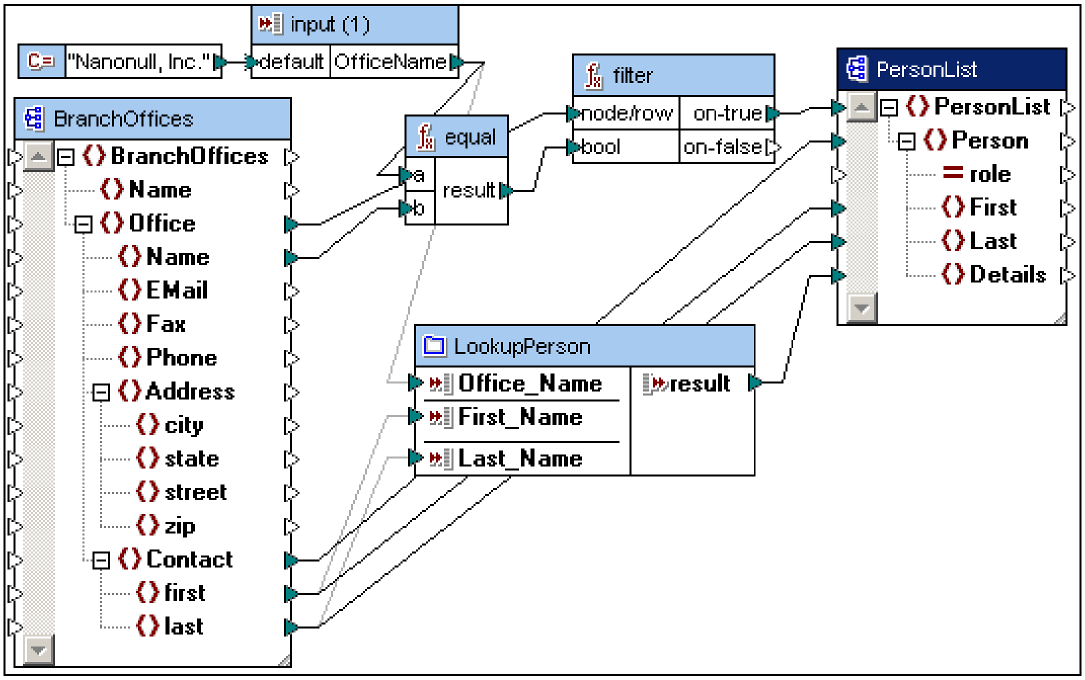


```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
  <LineItems>
    <LineItem>
      <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <SinglePrice>34</SinglePrice>
        <Price>102</Price>
      </Article>
    </LineItem>
    <LineItem>
      <Article>
        <Number>1</Number>
        <Name>T-Shirt</Name>
        <SinglePrice>25</SinglePrice>
        <Price>25</Price>
      </Article>
```

10.5 User-defined function - example

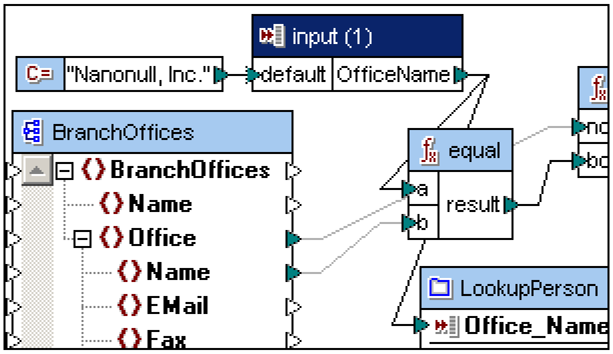
The PersonByListBranchOffice.mfd file available in the ...\MapForceExamples folder, describes the following features in greater detail:

- Nested User-defined functions e.g. **LookupPerson**
- Look-up functions that generate a string output e.g. **LookupPerson**
- **Optional** input-parameters which can also supply a **default** value e.g. the **EqualAnd** component (contained in the LookupPerson component)
- **Configurable** input parameters, which can also double as a command line parameter(s) when executing the generated mapping code!



Configurable input parameters
The input (1) component receives data supplied when a mapping is executed. This is possible in two ways:

- as a command line parameter when executing the generated code, e.g. Mapping.exe /OfficeName "Nanonull Partners, Inc."
- as a preview value when using the MapForce Engine to preview the data in the Output window.

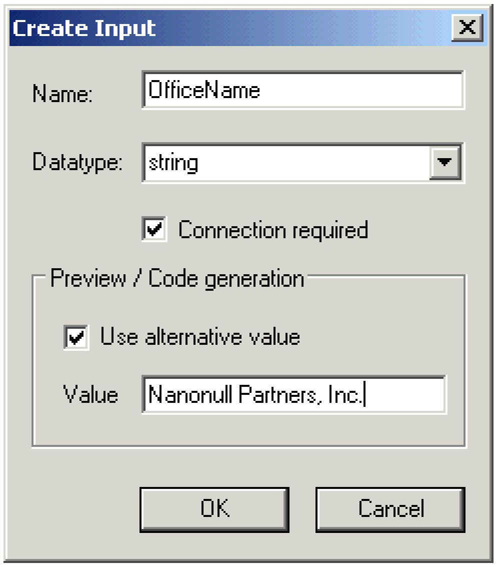


To define the Input value:

1. Double click the input (1), component and enter a different value in the "Value" text box of the Preview Mode group e.g. "Nanonull Partners, Inc.", and click OK to confirm.
2. Click the Output tab to see the effect.
   A different set of persons are now displayed.

   Please note that the data entered here is only used in "preview" mode i.e. when clicking the Output tab. If a value is not entered, or the check box is deactivated, then the data mapped to the input icon "default" is used.

   Please see Input values, overrides and command line parameters for more information.



Create Input

Name: OfficeName

Datatype: string

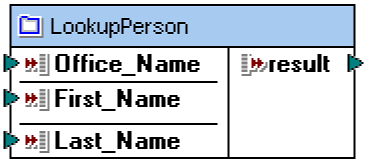
☑ Connection required

Preview / Code generation

☑ Use alternative value

Value Nanonull Partners, Inc.

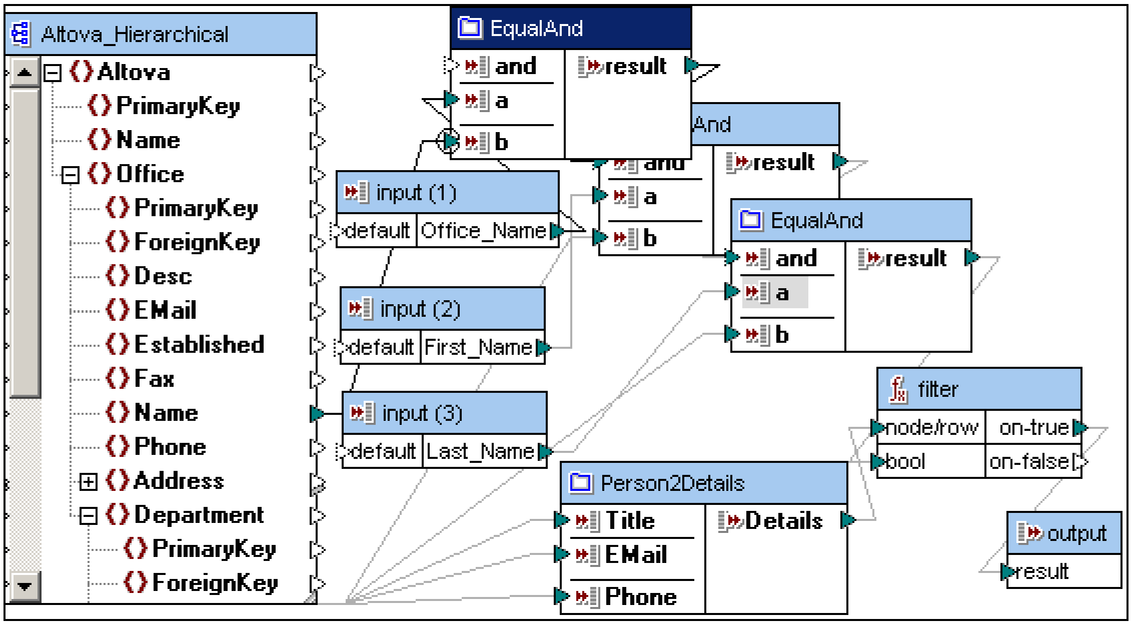
LookupPerson component



Double clicking this user-defined component displays its constituent components shown below. What this component does is:

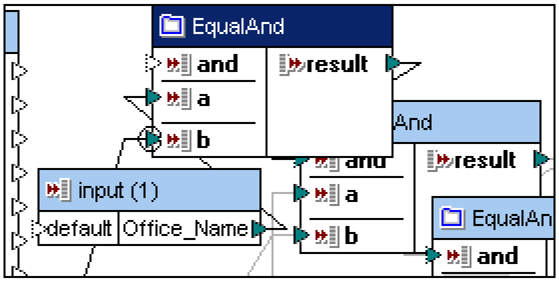
- Compares the Office, First, and Last names of BranchOffices.xml, with the same fields of the Altova_Hierarchical.xml file, using the **input** components and the **EqualAnd** user-defined components.
- Combines the Email, PhoneExt and Title items using the **Person2Details** user-defined function
- Passes on the combined person data to the **output** component if the previous EqualAnd comparisons are all true (i.e. supplied "true" to the filter component).

A user-defined function always outputs a value, which may even be an empty string! This would be the case if the filter component bool value is false. Only an empty string would be output instead of data supplied by the Person2Details component.



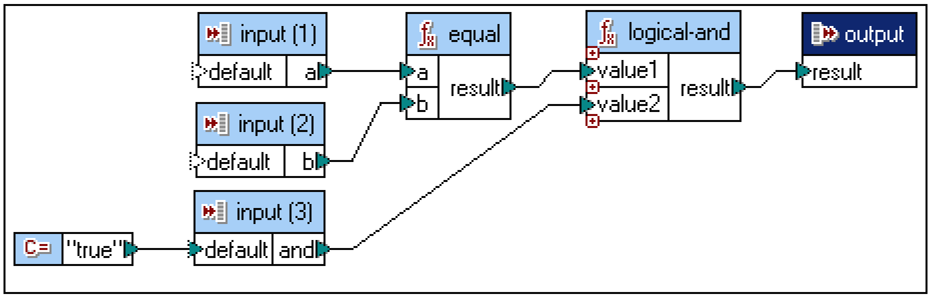
- The three **input** components, input (1) to input (3), receive their data from the BranchOffices.xml file.
- The **EqualAnd** component compares two values and provides an **optional** comparison value, as well as a default value.
- Person2Details combines three person fields and passes on the result to the filter component.

EqualAnd component



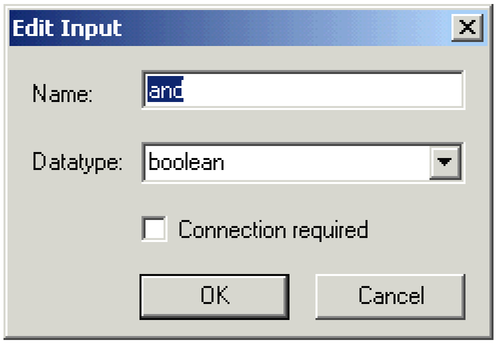
Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Compare two input parameters **a** and **b**, and pass the result on to the logical-and component. Note that the **b** parameter has been defined as the **priority context** (right click the icon to do so). This ensures that the person data of the specific office, supplied by the input parameter **a**, is processed first.
- **Logical-and** the result of the first comparison, with an **optional** input parameter, Input (3)
- Pass on the boolean value of this comparison to the output parameter.



Optional parameters

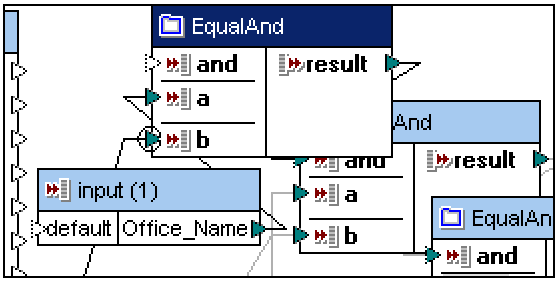
Double clicking the "input (3)" parameter, of the EqualAnd user-defined function shown above, allows you to make parameters optional, by unchecking the "Connection required" check box.


Edit Input

Name: and

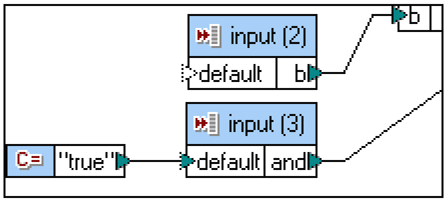
Datatype: boolean

□ Connection required



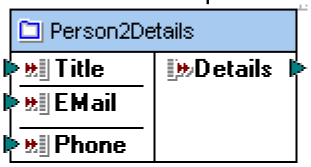
If "Connection required" is **unchecked**, then:

- A mapping connector is not required for the input icon of this user-defined function, e.g. the **and** parameter of the first EqualAnd function, does not have an input connector. The input icon has a dashed outline to show this visually.
- A **default** value can be supplied by connecting a component, within the user-defined function e.g. using a constant component containing the value "true".



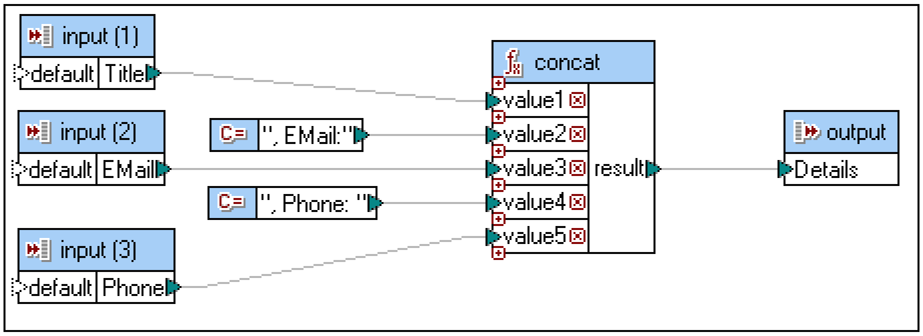
- A mapping from another item, mapped to the optional Input, takes precedence over the default value. E.g. the "and" parameter of second EqualAnd function, receives input data from the "result" parameter of the first EqualAnd user-defined function.

Person2Details component



Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Concatenate three inputs and pass on the result string to the output parameter.
- Double clicking an output parameter allows you to change the parameter name (Details), and select the datatype (String).



Chapter 11

Adding custom libraries

11 Adding custom libraries

MapForce allows you to create and add your own user-defined function libraries for Java, C# and C++.

Libraries can be added by clicking the Add libraries button under the Libraries pane, or by selecting the menu option **Tools | Options | Add** of the Libraries tab. The libraries are shown as files with an .mff extension.

Please note:
Mappings (data) using these types of user-defined functions cannot be previewed by clicking the Output tab, i.e. using the MapForce Engine, as they cannot be compiled by the MapForce engine. These functions are of course available when generating code!

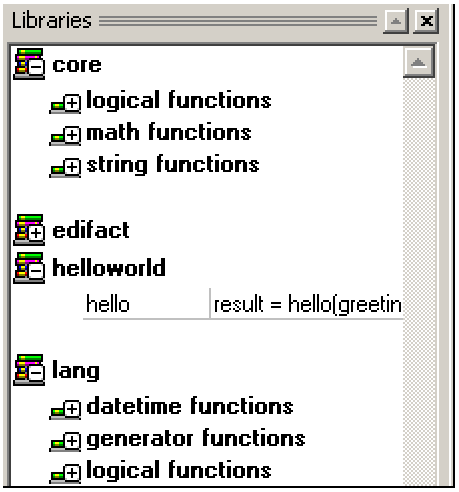
To be able to add user defined functions, you need:

- the mff file which tells MapForce what the interfaces to the functions are, and
- where the implementation can be found for the generated code.

A basic mff file for C# would for example look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd" version="2"
library="helloworld">
 <implementations>
  <implementation language="cs">
   <setting name="namespace" value="HelloWorldLibrary"/>
   <setting name="class" value="Greetings"/>
   <setting name="reference" value="C:\HelloWorldLibrary\
HelloWorldLibrary.dll"/>
  </implementation>
 </implementations>
 <group name="string functions">
  <component name="hello">
   <sources>
    <datapoint name="greeting_type" datatype="boolean"/>
   </sources>
   <targets>
    <datapoint name="result" datatype="string"/>
   </targets>
   <implementations>
    <implementation language="cs">
     <function name="HelloFunction"/>
    </implementation>
   </implementations>
   <description>
    <short>result = hello(greeting_type)</short>
    <long>Returns a greeting sentence according to the given
greeting_type.</long>
   </description>
  </component>
 </group>
</mapping>
```

The image below, shows the appearance of the mff file in MapForce. The new library "helloworld" appears as a library entry (sorted alphabetically), containing the "hello" string function.



Mff files can, of course, be written for more than one language. Every additional language must therefore contain an additional <implementation> element. The specifics on the implementation element are discussed later in this document.

11.1 Configuring the mff file

The steps needed to adapt the mff file to suit your needs, are described below.

The Library Name:
The library name is found in the mff file line shown below. Please make sure that the **library name** is written in **lowercase** letters.

```
<mapping  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="mff.xsd"
          version="2" library="helloworld">
```

The entry that will appear in the libraries window will be called "helloworld". Note that the library will **not** appear **immediately** after you have clicked the Add button in the Settings dialog box. Libraries are only displayed, if at least one component exists containing an implementation for an enabled programming Language.

Libraries and their functions can be toggled on or off, by deleting or adding the respective library file (*.mff).

To add the new mff file to the libraries pane:

1. Click the "Add libraries" button.
2. Click the "Add" button in the libraries dialog box.
3. Select the *.MFF library you want to include, and click Open to load the file in the Options dialog box.

Implementations Element for the helloworld library:

```
...
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mff.xsd" version="2"
library="helloworld">
 <implementations>
  <implementation language="cs">
   <setting name="namespace" value="HelloWorldLibrary"/>
   <setting name="class" value="Greetings"/>
   <setting name="reference" value="C:\HelloWorldLibrary\
HelloWorldLibrary.dll"/>
  </implementation>
 </implementations>
...
```

For each language that the helloworld library should support, an implementations element has to be added. The settings within each implementation, allow the generated code to call the specific functions defined in Java, c++ or c#.

The specific settings for each programming language will be discussed below.

Java:

```
...
<implementation language="java">
 <setting name="package" value="com.hello.functions"/>
 <setting name="class" value="Hello"/>
</implementation>
...
```

It is important for the generated code to be able to find your **hello.class** file. This can be achieved by making sure that it is entered in the classpath. The classpath is found in the system environment variables.

C#:

```
...
 <implementation language="cs">
  <setting name="namespace" value="HelloWorldLibrary"/>
  <setting name="class" value="Hello"/>
  <setting name="reference" value=" C:\HelloWorldLibrary\
HelloWorldLibrary.dll "/>
</implementation>
...
```

Note for C# : it is very important that the generated code uses the namespace which is defined here. C# also needs to know the location of the **dll** that is to be linked to the generated code.

C++:

```
...
<implementation language="cpp">
   <setting name="namespace" value="helloworld"/>
   <setting name="class" value="Greetings"/>
   <setting name="path" value="C:\HelloWorldLibrary"/>
   <setting name="include" value="Greetings.h"/>
   <setting name="source" value="Greetings.cpp"/>
  </implementation>
...
```

- **namespace** is the namespace in which your **Greetings** class will be defined. It must be equal to the library name.
- **path** is the path in which the include and the source files are to be found.
- The source files will then be copied to the directory **targetdir/libraryname** (defined when selecting the menu option **File | Generate xxx code**, and selecting the directory).
- If you have multiple include files or source files, just add an additional **setting** element for include or source.

All the include files you supply will be included in the generated Algorithm.

Adding a component:
Each component you will define, will be located within a function group. Staying with the helloworld example:

```
...
<group name="string functions">
 <component name="hello">
  …
 </component>
</group>
...
```

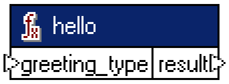
Please make sure the component name (hello) is in lowercase, or it will not appear in the library window.

11.2 Defining the component user interface

The code shown below, defines how the component will appear when dragged into the mapping area.

```
...
<component name="hello">
 <sources>
  <datapoint name="greeting_type" datatype="boolean"/>
 </sources>
 <targets>
  <datapoint name="result" datatype="string"/>
 </targets>
 <implementations>
 …
 </implementations>
 <description>
  <short>result = hello(greeting_type)</short>
  <long>Returns a greeting sentence according to the given
greeting_type.</long>
 </description>
</component>
...
```

The new MapForce component:



Datapoints
Datapoints can be loosely defined as the input, or output parameters of a function. The datapoints datatype parameter, specifies the parameters/return values, type.

Please note:
Only one **target** datapoint, but multiple **source** datapoints are allowed for each function.

The datatype of each datapoint, must be one of the following datatypes:

- anyType
- boolean
- decimal
- string
- datetime
- duration
- date
- time

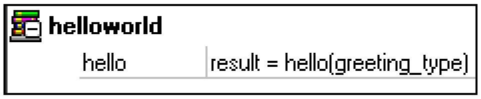
These datatypes have to correspond to the datatypes of the function's parameters you defined in your Java, C++ or C# library.

Altova has therefore provided support for Schema simpleTypes as classes, for each of the supported programming languages. The integration of these Schema simpleTypes in your library, will be explained later in this document.

Function Descriptions:
Functions are accompanied by short and long descriptions in the library window. The short description is always shown to the right of the function name, while the long description is

displayed as a ToolTip when you place the mouse cursor over the short description.

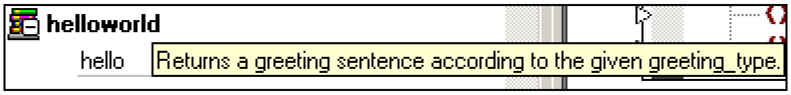
Short description:



helloworld

hello result = hello(greeting_type)

Long description:



helloworld

hello Returns a greeting sentence according to the given greeting_type.

11.3 Function implementation details

We are now at the point where we need to make a connection between the function in the library window, and the function in the Java, C# or C++ classes. This is achieved with the <implementation> element.

As previously stated, one function may have multiple implementation elements – one for each supported programming languages.

```
...
<component name="hello">
...
 <implementations>
  <implementation language="cs">
   <function name="HelloFunction"/>
  </implementation>
 </implementations>
...
</component>
...
```

A function may be called "HelloFunction" in Java, or "HelloFunctionResponse" in C++. This is why you need to specify a separate function name for each programming language.

A function for each of the three programming languages might look like the following:

```
...
<component name="hello">
...
 <implementations>
  <implementation language="cs">
   <function name="HelloFunction"/>
  </implementation>
  <implementation language="java">
   <function name="HelloFunction"/>
  </implementation>
  <implementation language="cpp">
   <function name="HelloFunctionResponse"/>
  </implementation>
 </implementations>
...
</component>
...
```

The value you supply as function name, must of course, exactly match the name of the function in the Java, C# or C++ class.

11.4 Writing your libraries

How to write a Java library:

1. Create a new Java class, using the previous example, name it "Hello".
2. Add the package name you provided under

```
...
<implementation language="java">
            <setting name="package"
value="com.hello.functions"/>
            <setting name="class" value="Hello"/>
</implementation>
...
```

3. Add the line "import com.altova.types.*;" – this enables you to use the Schema simpleTypes.

   If you encounter problems finding the **com.altova.types** on your computer, please generate and compile Java code without user defined functions; you will then find the classes in the directory you specified.

4. Add the functions you specified in the mff file as public static.
   Do not forget that you may only use the Schema simpleTypes as input/output parameters!

```
package com.hello.functions;
import com.altova.types.*;

public class Hello {
      public static SchemaString HelloFunction ( SchemaBoolean
GreetingType ) {
            if( GreetingType.getValue() )
                  return new SchemaString("Hello World!");
            return new SchemaString("Hello User!");
      }
}
```

5. Compile the Java file to a class file, and add this to your Classpath. You have now finished creating your custom library.

How to write a C# library:

1. Open a new Project and create a class library
2. Go to add reference, and add the **Altova.dll**

   If you encounter problems finding **Altova.dll** on your computer, please generate and compile the C# code without user defined functions; you will then find the DLL in the directory you specified.

3. Add the "**using Altova.Types;**" line
4. The class name should be the same as you specified (here "Greetings")

```
  <implementation language="cs">
      <setting name="namespace" value="HelloWorldLibrary"/>
      <setting name="class" value="Greetings"/>
      <setting name="reference" value="C:\HelloWorldLibrary\
HelloWorldLibrary.dll"/>
  </implementation>
```

5. Add the namespace using the same value as you specified in the implementation

settings shown above

6. Add your functions as **public static**, and remember to only use the Schema simpleTypes supplied by the Altova.dll

The sample code should look like this:

```
 using System;
 using Altova.Types;

 namespace HelloWorldLibrary
 {
   public class Greetings
   {
     public Greetings()
     {
     }

     public static SchemaString HelloFunction(SchemaBoolean
GreetingType)
     {
       if( GreetingType.Value )
         return new SchemaString("Hello World!");
       return new SchemaString("Hello User!");
     }
   }
 }
```

7. The last step is to compile the code.
   The path where the compiled dll is located, must match the "reference" setting in the implementation element.

How to write a C++ library:

Create the **header** and **cpp** files using the exact name, at the same location you defined in the implementation element, for the whole library.

Header file:

1. Write "using namespace altova;"
2. Add the namespace you specified in the implementation element.
3. Add the class you specified in the implementation element of the mff, with the functions you specified in the mff.
4. Please remember to write "ALTOVA_DECLSPECIFIER" in front of the class name, this ensures that your classes will compile correctly - whether you use dynamic or static linkage in subsequent generated code.
5. Remember to use only the schema simpleTypes as input parameters and return values (defined in schematypes.h of generated C++ code, which start with CSchemaType….)

The resulting **header file** should look like this:

```
#ifndef HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
#define HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED

#if _MSC_VER > 1000
   #pragma once
#endif // _MSC_VER > 1000

using namespace altova;

namespace helloworldlibrary {
```

```
class ALTOVA_DECLSPECIFIER Greetings
{
public:
   static CSchemaString    HelloFunctionResponse(const
   CSchemaBoolean& rGreetingType);
};

} // namespace HelloWorldLibrary

#endif // HELLOWORLDLIBRARY_GREETINGS_H_INCLUDED
```

In the **cpp** file:

1. The first lines need to be the includes for **StdAfx.h** and the definitions from the **Altova** base library, please copy these lines from the sample code supplied below.
2. The **../Altova** path is correct for your source files, because they will be copied to a separate project in the resulting code that will be found at targetdir/libraryname.
3. The next line is the include for your header file you created above.
4. Add the implementations for your functions.
5. Please remember that the implementations need to be in the correct namespace you specified in the header file and in the implementations element of the mff.

The sample cpp file would look like this:

```
#include "StdAfx.h"
#include "../Altova/Altova.h"
#include "../Altova/AltovaException.h"
#include "../Altova/SchemaTypes.h"
#include "../Altova/SchemaTypeString.h"

#include "Greetings.h"

namespace helloworld {

   CSchemaString     Greetings::HelloFunctionResponse(const
   CSchemaBoolean& rGreetingType)
   {
         if( rGreetingType )
               return CSchemaString( _T("Hello World!") );
         return CSchemaString( _T("Hello User!") );
   }

}
```

In contrast to Java or C#, you do not need to compile your source files. They will be copied to the generated code, and are compiled with the rest of the generated mapping code.

C++ compile errors:

If you get a compiler error at the line shown below, add the path to the msado15.DLL

```
#import "msado15.dll" rename("EOF", "EndOfFile")
```

You have to add the path where the msado15.dll is stored into the directories section of your Visual Studio environment:

1. In VS select from the menu: Tools / Options...
2. Select the "Directories" tab.
3. Select "Include files" in the pull-down "Show directories for"
4. Add a new line with the path to the file;

for English systems usually "C:\Program Files\Common Files\System\ADO"

5. Rebuild, then everything should be fine.

Chapter 12

Adding custom XSLT 1.0 functions

12 Adding custom XSLT 1.0 functions

MapForce allows you to extend the installed XSLT function libraries with your own custom functions. This option is made available when you select XSLT as the output, by clicking the XSLT icon, or selecting **Output | XSLT 1.0**.

XSLT files appear as libraries, and display all **named templates** as functions below the library name.

- Functions must be declared as Named Templates conforming to the XSLT 1.0 specification in the XSLT file.
- If the imported XSLT file imports, or includes other XSLT files, then these XSLT files and functions will be imported as well.
- Each named template appears as a function below each library name.
- The amount of mappable input icons, depends on the number of parameters used in the template call; optional parameters are also supported.
- Updates to imported XSLT files, occur at program start.
- Namespaces are supported

Please note:

When writing named templates please make sure that the XPath statements used in the template are bound to the correct namespace(s). The namespace bindings of the mapping can be viewed by clicking the XSLT tab. Please see: the XSLT 1.0 implementation specific document for more information.

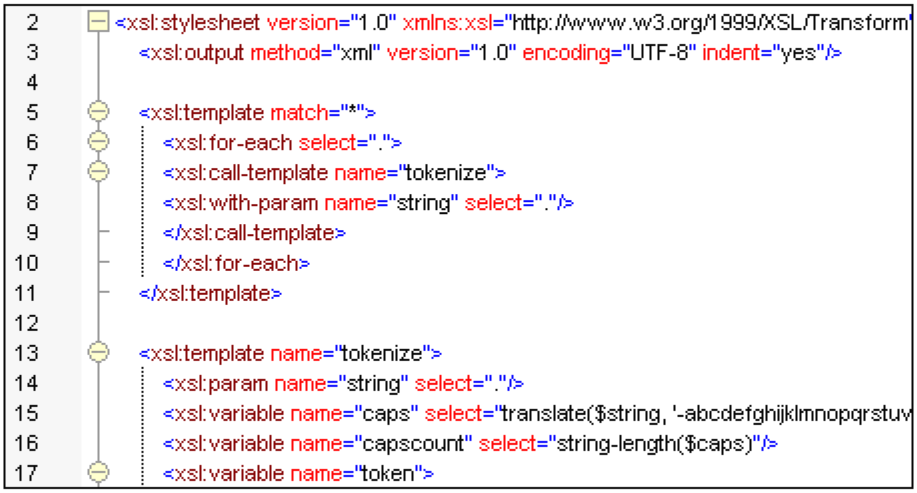
The files needed for the simple example shown below, are available in the ...\ **MapForceExamples** directory.

- Name-splitter.xslt
- Name-splitter.xml (the XML instance file for Customer.xsd)
- Customers.xsd
- CompletePO.xsd

Please see: Aggregate functions for an additional example of using named templates to sum nodes.

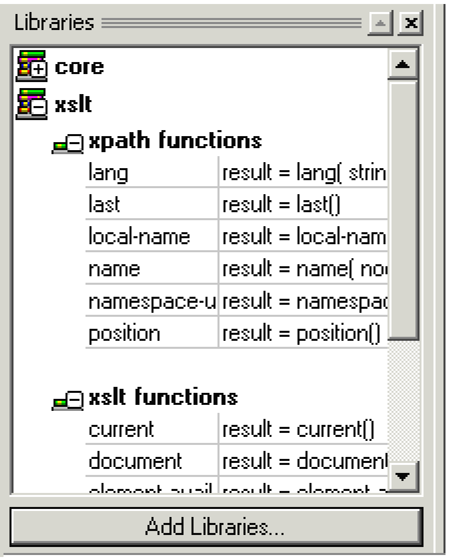
To add a custom XSLT function:

1. Create an XSLT file that achieves the transformation/result you want.
   The example below, **Name-splitter.xslt**, shows a named template called "**tokenize**" with a single parameter "string". What the template does, is work through an input string and separate capitalized characters with a space for each occurrence.



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="*">
        <xsl:for-each select=".">
        <xsl:call-template name="tokenize">
        <xsl:with-param name="string" select="."/>
        </xsl:call-template>
        </xsl:for-each>
    </xsl:template>

    <xsl:template name="tokenize">
        <xsl:param name="string" select="."/>
        <xsl:variable name="caps" select="translate($string, '-abcdefghijklmnopqrstuv
        <xsl:variable name="capscount" select="string-length($caps)"/>
        <xsl:variable name="token">
```

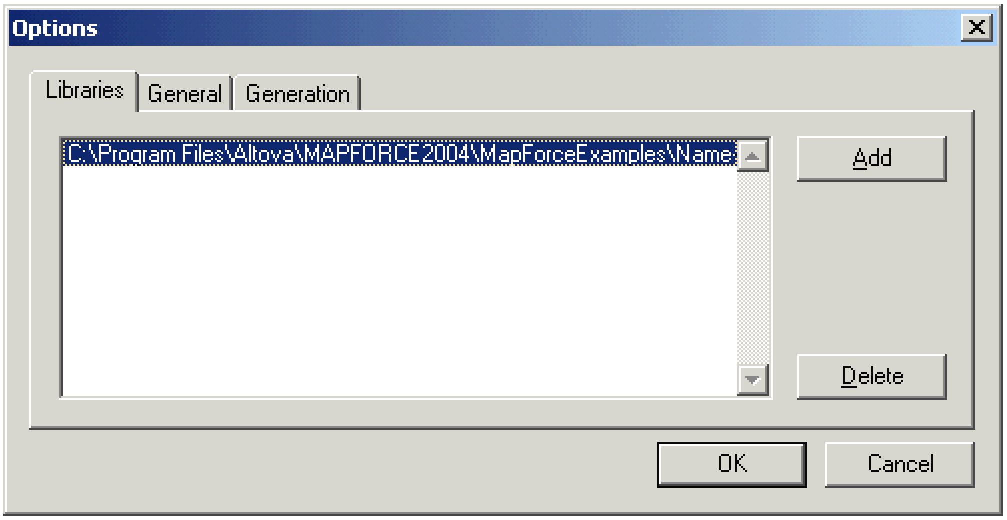
2. Click the **Add Libraries** button, and then click the Add button in the following dialog box.



lang	result = lang(strin
last	result = last()
local-name	result = local-nam
name	result = name(no
namespace-u	result = namespac
position	result = position()

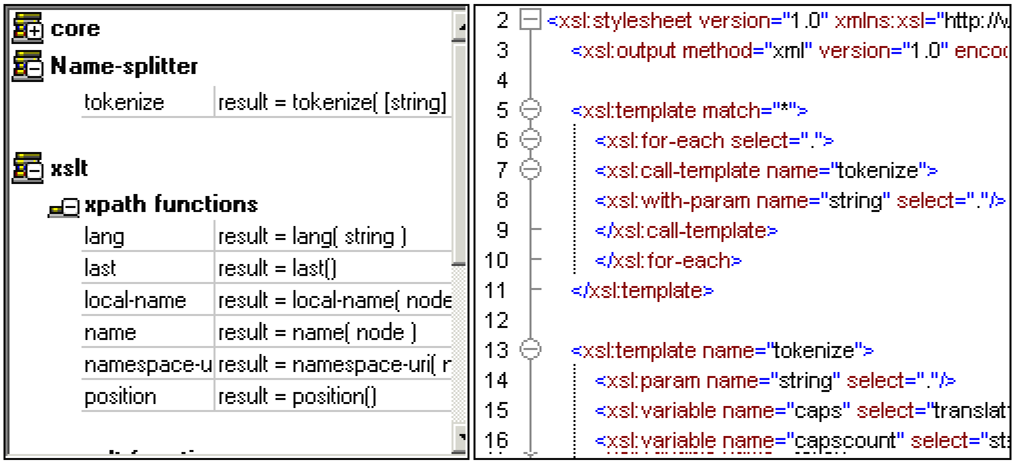
current	result = current()
document	result = document

3. Select the XSL, or XSLT file, that contains the named template you want to act as a function, in this case **Name-splitter.xslt**. The XSLT file appears in the Libraries tab.



C:\Program Files\Altova\MAPFORCE2004\MapForceExamples\Name

4. Click **OK** to insert the new function.

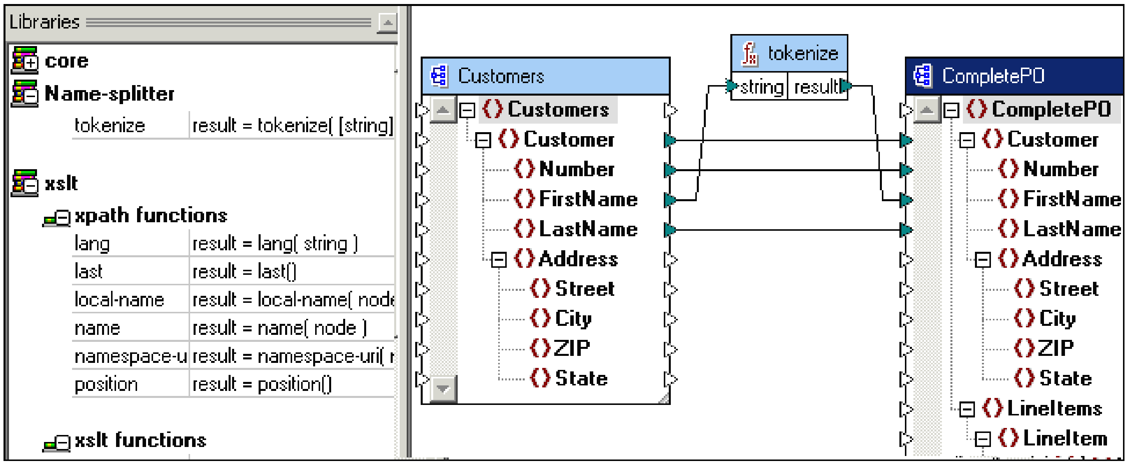


core	
Name-splitter	
tokenize	result = tokenize([string]
xslt	
xpath functions	
lang	result = lang(string)
last	result = last()
local-name	result = local-name(node
name	result = name(node)
namespace-u	result = namespace-uri(r
position	result = position()

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://w
    <xsl:output method="xml" version="1.0" encod

    <xsl:template match="*">
        <xsl:for-each select=".">
        <xsl:call-template name="tokenize">
        <xsl:with-param name="string" select="."/>
        </xsl:call-template>
        </xsl:for-each>
    </xsl:template>

    <xsl:template name="tokenize">
        <xsl:param name="string" select="."/>
        <xsl:variable name="caps" select="translat
        <xsl:variable name="capscount" select="st
```

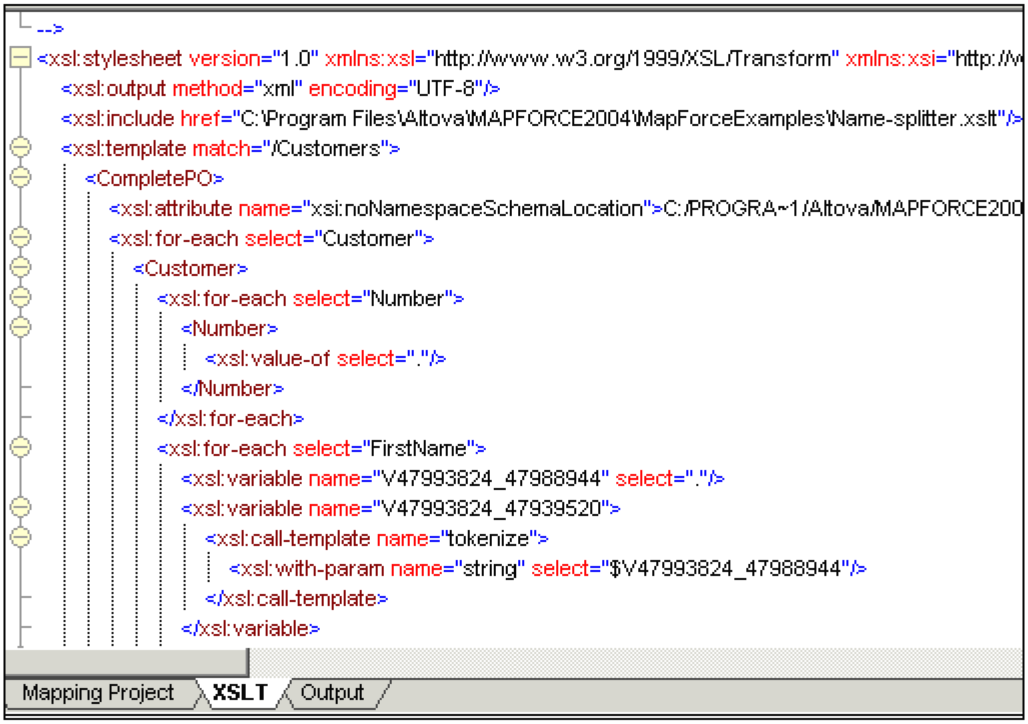
The XSLT file name appears in the library window, along with the function(s) defined as named templates, below it. In this example **Name-splitter** with the **tokenize** function.

5. Drag the function into the Mapping window, to use it in you current mapping, and map the necessary items, as show in the screenshot below.



core	
Name-splitter	
tokenize	result = tokenize([string]
xslt	
xpath functions	
lang	result = lang(string)
last	result = last()
local-name	result = local-name(node
name	result = name(node)
namespace-u	result = namespace-uri(r
position	result = position()
xslt functions	

6. Click the XSLT tab to see the generated XSLT code.

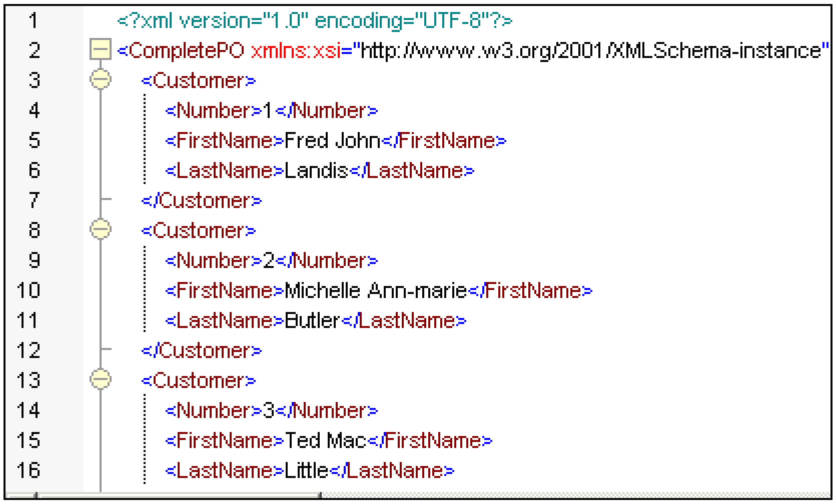


```
-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xsi="http://w
    <xsl:output method="xml" encoding="UTF-8"/>
    <xsl:include href="C:\Program Files\Altova\MAPFORCE2004\MapForceExamples\Name-splitter.xslt"/>
    <xsl:template match="/Customers">
        <CompletePO>
            <xsl:attribute name="xsi:noNamespaceSchemaLocation">C:/PROGRA~1/Altova/MAPFORCE200
            <xsl:for-each select="Customer">
                <Customer>
                    <xsl:for-each select="Number">
                        <Number>
                            <xsl:value-of select="."/>
                        </Number>
                    </xsl:for-each>
                    <xsl:for-each select="FirstName">
                        <xsl:variable name="V47993824_47988944" select="."/>
                        <xsl:variable name="V47993824_47939520">
                            <xsl:call-template name="tokenize">
                                <xsl:with-param name="string" select="$V47993824_47988944"/>
                            </xsl:call-template>
                        </xsl:variable>
```

Please note:

As soon as a named template is used in a mapping, the XSLT file containing the named template is **included** in the generated XSLT code (**xsl:include href...**), and is **called** using the command **xsl:call-template**.

7. Click the Output tab to see the result of the mapping.



```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Customer>
    <Number>1</Number>
    <FirstName>Fred John</FirstName>
    <LastName>Landis</LastName>
  </Customer>
  <Customer>
    <Number>2</Number>
    <FirstName>Michelle Ann-marie</FirstName>
    <LastName>Butler</LastName>
  </Customer>
  <Customer>
    <Number>3</Number>
    <FirstName>Ted Mac</FirstName>
    <LastName>Little</LastName>
```

To delete custom XSLT functions:

1. Click the **Add Libraries** button.
2. Click to the specific XSLT **library name** in the Libraries tab
3. Click the Delete button, then click OK to confirm.

Chapter 13

Adding custom XSLT 2.0 functions

13 Adding custom XSLT 2.0 functions

MapForce also allows you to import XSLT 2.0 functions that occur in an XSLT 2.0 document in the form:

<xsl:function name="MyFunction">

Please see: the XSLT 2.0 implementation specific document for more information, as well as Aggregate functions for an additional example of using named templates to sum nodes.

Datatypes in XPath 2.0

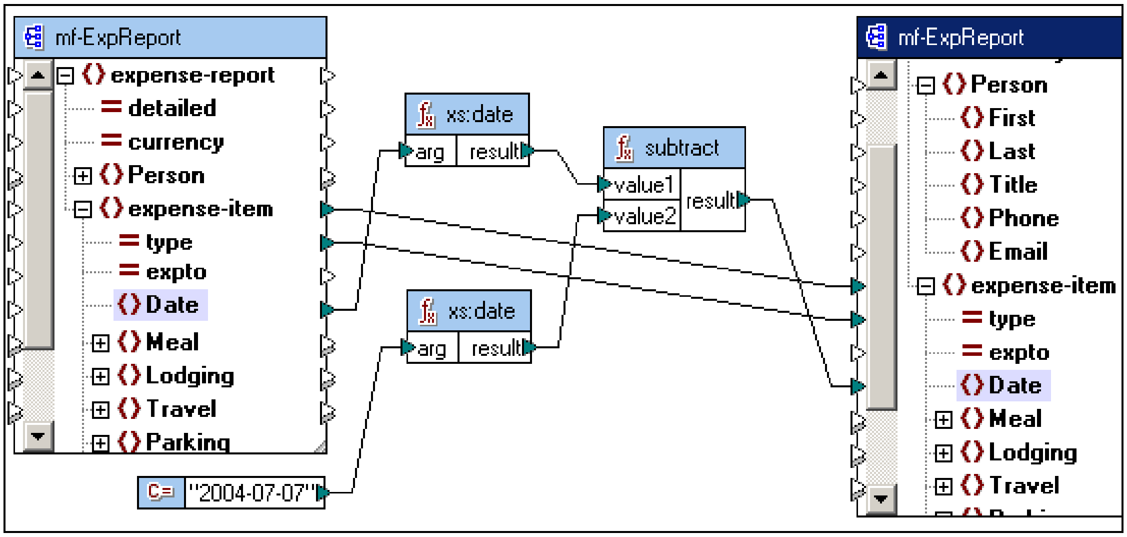
If your XML document references an XML Schema and is valid according to it, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation.

In the XPath 2.0 Data Model used by the Altova XSLT 2.0 Engine, all **atomized** node values from the XML document are assigned the `xdt:untypedAtomic` datatype. The `xdt:untypedAtomic` type works well with implicit type conversions.

For example,

- the expression `xdt:untypedAtomic("1") + 1` results in a value of 2 because the `xdt:untypedAtomic` value is **implicitly** promoted to `xs:double` by the addition operator.
- Arithmetic operators implicitly promote operands to `xs:double`.
- Value comparison operators promote operands to `xs:string` before comparing.

In some cases, however, it is necessary to **explicitly** convert to the required datatype.



To allow date calculations, as shown in the example above, the value of the Date element, although defined as being of type xs:date datatype, must be explicitly converted to xs:date (using the xs:date constructor) before it can be used for a date calculation.

Similarly, the string constant "2004-07-07" must also be explicitly converted to the xs:date datatype before being used for a date calculation.

The subtract function when performed on two xs:date values, is actually the abstract op:subtract-dates function which returns an xdt:dayTimeDuration value.

Chapter 14

Adding custom XQuery functions

14 Adding custom XQuery functions

MapForce allows you to import XQuery library modules.

Please see: the XQuery implementation specific document for more information.

Chapter 15

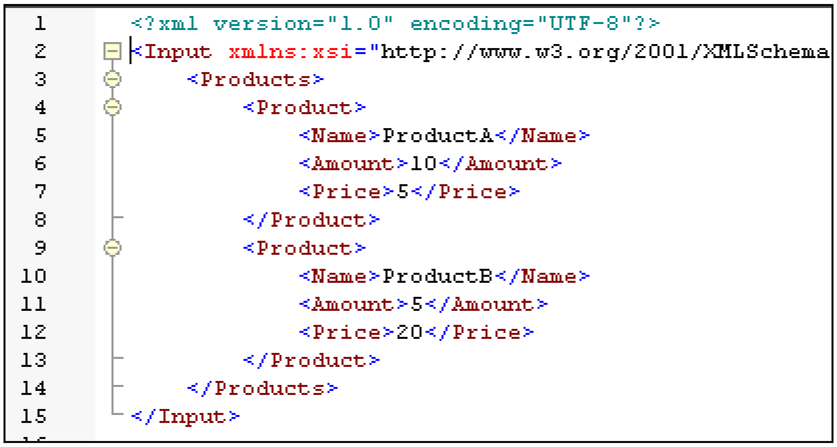
Aggregate functions - summing nodes in XSLT1 and 2

15 Aggregate functions - summing nodes in XSLT1 and 2

This section describes the method you can use to process multiple nodes of an XML instance document and have the result mapped as a single value to a target item. The files used in this example are available in the ...\MapforceExamples\Tutorial folder and consists of:

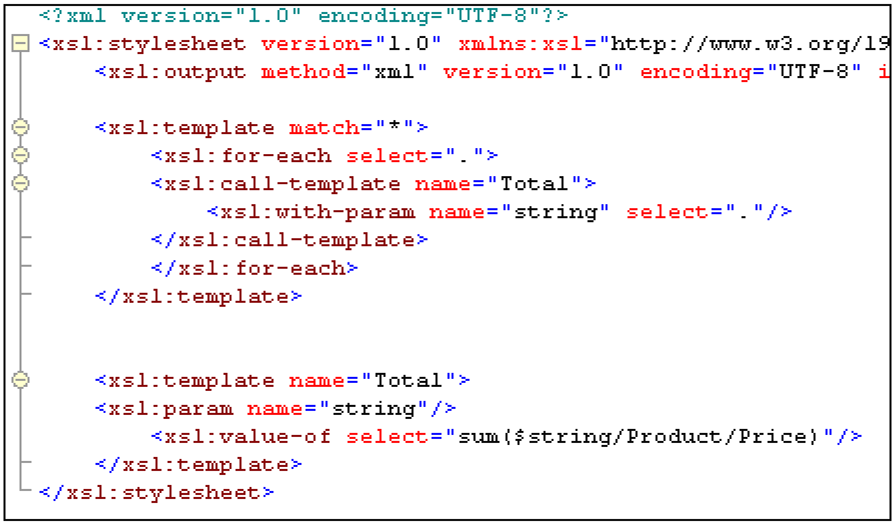
Summing-nodes.mfd	mapping
input.xml	input XML file
input.xsd and output.xsd	source and target schema files
Summing-nodes.xslt	xslt file containing a named template to sum the individual nodes

The screenshot below shows the **XML input** file. The aim of the example is to sum the Price fields of any number of products, in this case products A and B.



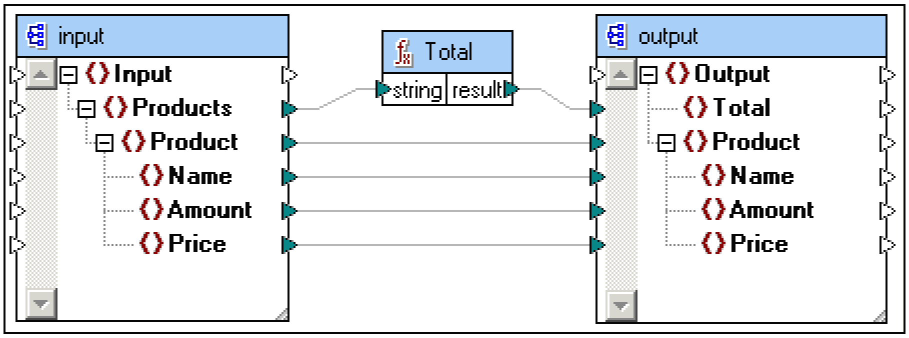
```
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema
    <Products>
        <Product>
            <Name>ProductA</Name>
            <Amount>10</Amount>
            <Price>5</Price>
        </Product>
        <Product>
            <Name>ProductB</Name>
            <Amount>5</Amount>
            <Price>20</Price>
        </Product>
    </Products>
</Input>
```

The screenshot below shows the XSLT stylesheet which uses the named template "**Total**" and a single parameter "string". What the template does, is work through the XML input file and sum all the values obtained by the XPath expression **/Product/Price**, in the document.

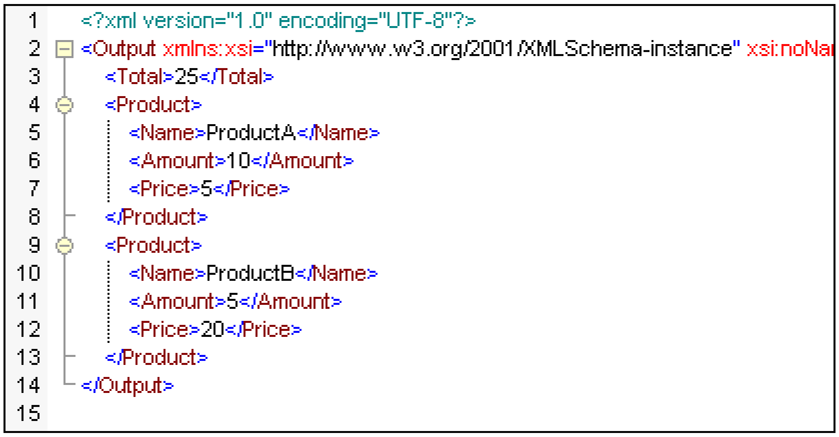


```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/19
    <xsl:output method="xml" version="1.0" encoding="UTF-8" i

    <xsl:template match="*">
        <xsl:for-each select=".">
        <xsl:call-template name="Total">
            <xsl:with-param name="string" select="."/>
        </xsl:call-template>
        </xsl:for-each>
    </xsl:template>


    <xsl:template name="Total">
    <xsl:param name="string"/>
        <xsl:value-of select="sum($string/Product/Price)"/>
    </xsl:template>
</xsl:stylesheet>
```

1. Click the **Add Libraries** button, and select the Libraries tab of the Options dialog box.
2. Click the Add button and select the **Summing-nodes.xslt** file from the ...\MapforceExamples\Tutorial folder.
3. Drag in the Total function from the newly created Summing-nodes library and create the mappings as shown below.



4. Click the Output tab to preview the mapping result.

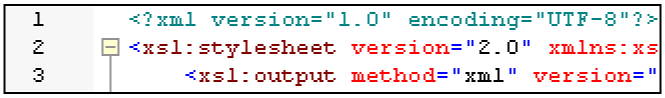


```
<?xml version="1.0" encoding="UTF-8"?>
<Output xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNa
  <Total>25</Total>
  <Product>
    <Name>ProductA</Name>
    <Amount>10</Amount>
    <Price>5</Price>
  </Product>
  <Product>
    <Name>ProductB</Name>
    <Amount>5</Amount>
    <Price>20</Price>
  </Product>
</Output>
```

The two Price fields of both products have been added and placed into the Total field.

To sum the nodes in XSLT 2.0:

- Change the stylesheet declaration in the template to ... version="2.0".



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xs
    <xsl:output method="xml" version="
```

Chapter 16

Type conversion checking

16 Type conversion checking

From MapForce 2006 **SP2** on, the generated applications and preview (with the builtin execution engine) check for type-conversion errors in more detail, inline with XSLT2 and XQUERY.

Converting values from one type to another may now result in a runtime-error, where in prior versions of MapForce would have produced some type of result.

Example: conversion of a xs:string 'Hello', to xs:decimal

MapForce 2006 Versions up to, and including **SP1**:

XSLT:	'Hello' (or 'NaN' when passed to a function dealing with number)
XSLT2:	error: "invalid lexical value"
Xquery:	error: "invalid lexical value"
Preview with BUILTIN-engine:	0
C++ app:	0
C# app:	error: "values not convertable"
Java app:	error: "values not convertable"

MapForce 2006 **SP2**:

XSLT:	'Hello' (or 'NaN' when passed to a function dealing with number)
XSLT2:	error: "invalid lexical value"
Xquery:	error: "invalid lexical value"
Preview with BUILTIN-engine:	**error: "string-value 'Hello' could not be converted to decimal"**
C++ app:	**error: "values not convertable"**
C# app:	error: "values not convertable"
Java app:	error: "values not convertable"

If type-conversion-errors occur, check that the types have been handled correctly. E.g. use the lang:numeric() function, to check if the source-value may be converted into a number, and then an if-else component to pass a different value in case it fails (e.g. a constant containing -1, on the value-false parameter).

Chapter 17

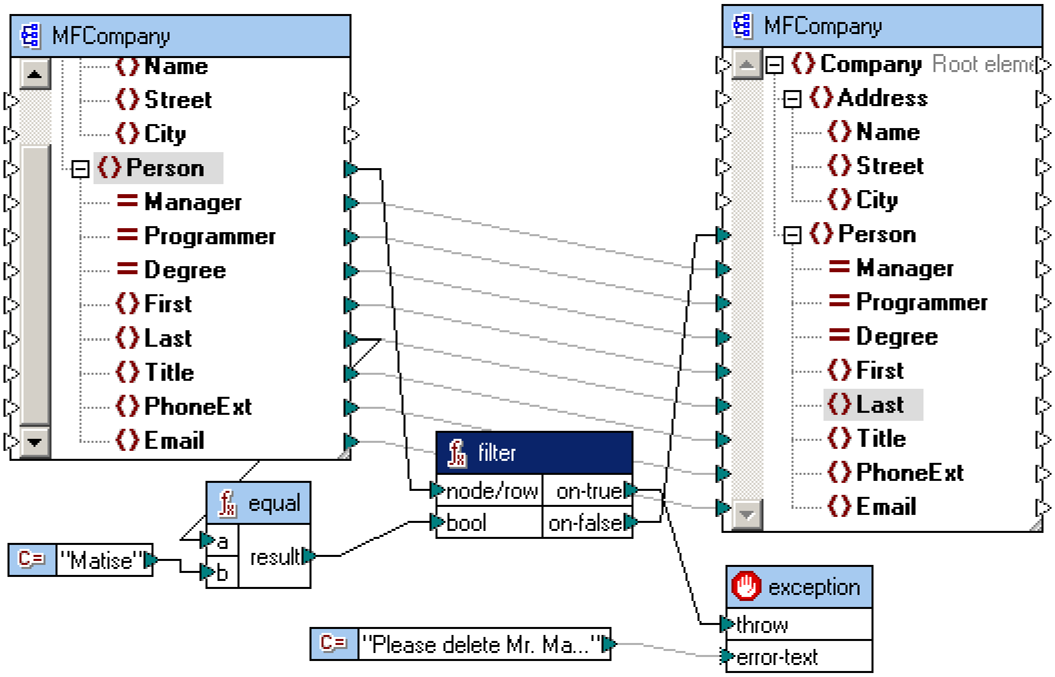
MapForce Exceptions

17 MapForce Exceptions

MapForce provides support for the definition of exceptions. You can define the condition that will throw an error. When the condition is satisfied, a user-defined message appears and the mapping process is stopped. The **ExpenseLimit.mfd** file in the MapForceExamples folder is a sample mapping that contains an exception function.

To insert an exception component:

- Select the menu option **Insert | Exception**, or click the Exception icon in the icon bar.



The example above shows how exceptions are defined in mappings. The exception should be triggered when the Last name of a person equals "Matise".

- The **equal** component checks to see if Last equals Matise, and the bool result is passed on to the filter component.
- When the condition is satisfied, i.e. Matise is **True**, the **on-true** parameter of the filter component activates the exception and the mapping process is stopped. (Note that you can also connect the exception to the on-false parameter, if that is what you need.)
- The error text supplied by the **constant** component is output.
- The error text appears in the Output tab, and also when running the compiled code.

Please note:
It is very important to note the filter placement in the example:

- **Both parameters** of the filter component, on-true and on-false, must be mapped! One of them needs to be mapped to the fault component, and the other, to the target component that receives the filtered source data. If this is not the case, the exception component will never be triggered.
- The **exception** and **target** components must be **directly connected** to the **filter** component. Functions, or other components, may not be placed between the filter and

either the exception, or target components.

- When generating **XSLT 2.0** and **XQuery** code, the exception appears in the Messages window, and a Preview failed message box appears. Clicking the OK button in the message box switches to the respective XSLT2 or XQuery tab, and the line that triggered the exception is automatically highlighted.

Chapter 18

MapForce engine

18 MapForce engine

The MapForce engine allows you to immediately preview and save the result of a transformation, without having to go the path of generating program code, compiling it, and viewing the results. This is achieved by simply clicking the **Output** tab. The Output tab also supports the find command, enabling you to find any XML data, or SQL statement that you might need to find.

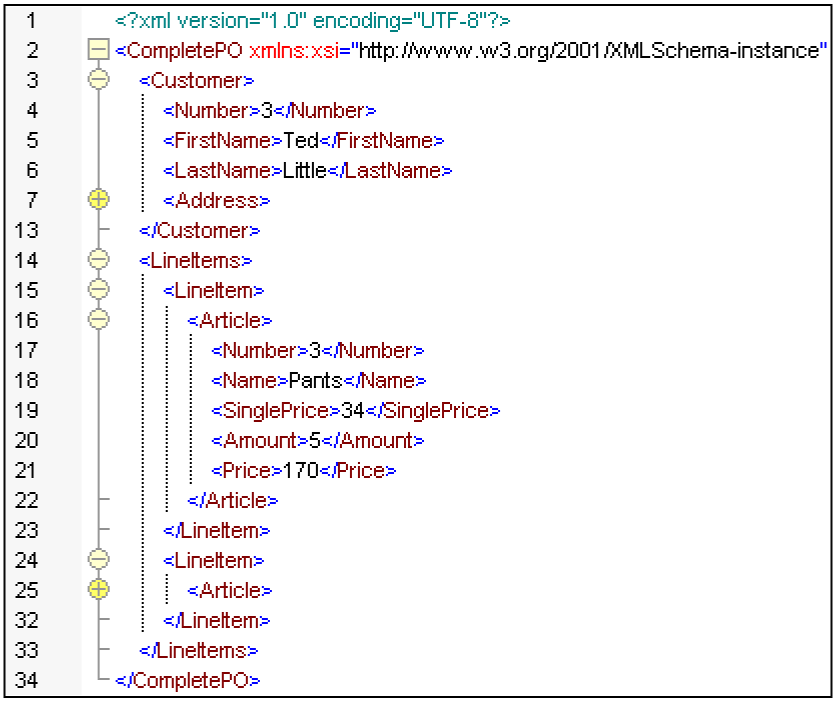
MapForce can also be started from the **command line** and produce a result, without having to generate any intermediate code. We would still, however, recommend that code be generated in one of the respective programming languages (after the development phase) due to the better execution speed of generated code.

Depending on the **target** component of your mapping, the Output tab may show different things:

XML Schema/document as target:

The result of the mapping is immediately presented in the Output tab. Any data source components can be used: XML/Schema files, Text and CSV files, databases; the result you would have achieved if the Java, C++, or C# code had been executed, appear in the Output tab.

The screen shot below, shows the output of the **DB_CompletePO.mfd** mapping available in ...\MapForceExamples folder. An XML Schema/document, as well as a database are used as source components in this mapping.

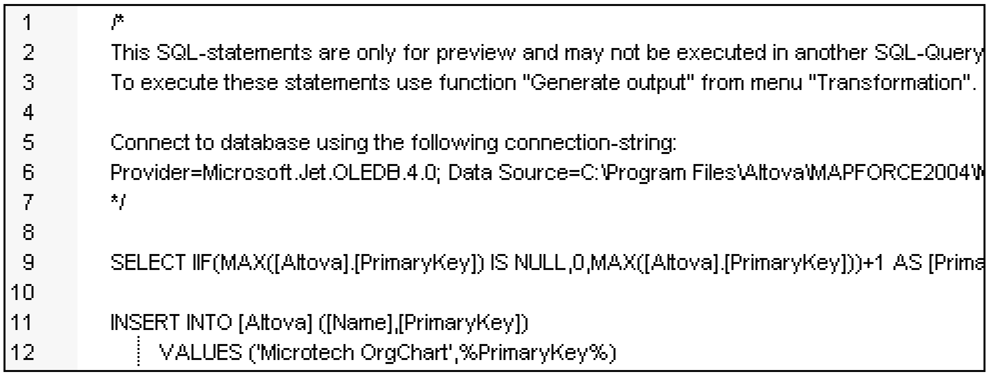


```
<?xml version="1.0" encoding="UTF-8"?>
<CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Customer>
    <Number>3</Number>
    <FirstName>Ted</FirstName>
    <LastName>Little</LastName>
    <Address>
  </Customer>
  <LineItems>
    <LineItem>
      <Article>
        <Number>3</Number>
        <Name>Pants</Name>
        <SinglePrice>34</SinglePrice>
        <Amount>5</Amount>
        <Price>170</Price>
      </Article>
    </LineItem>
    <LineItem>
      <Article>
    </LineItem>
  </LineItems>
</CompletePO>
```

The resultant XML file can be saved by clicking the Save icon, and validated against the referenced schema by clicking the validate icon in the icon bar.

Database as target:

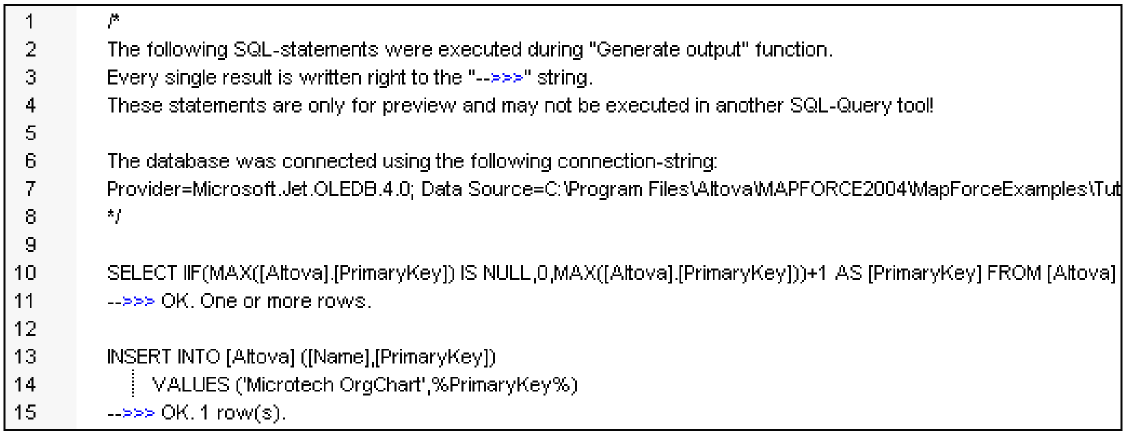
SQL pseudo-code is displayed when a database component is the target. The complete select statement, (all Select, Insert, Update or Delete statements), is displayed for you to preview before executing the SQL statement.



```
/*
This SQL-statements are only for preview and may not be executed in another SQL-Query
To execute these statements use function "Generate output" from menu "Transformation".

Connect to database using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MAPFORCE2004\
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [Prima

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
```

Clicking the Run SQL-script icon, executes the SQL select statement and presents you with a report on the database actions, as shown in the screen shot below.

- Actual SQL statement that were executed on the target database
- Multiple table actions if any occurred i.e. "UPDATE ….. -->>> OK. 0 rows affected." and the "INSERT …. -->> OK. 1 rows affected".
- Results of every SQL statement:
  e.g. **OK** and xx rows affected if successful, or **FAILED**, and a detailed error message.



```
/*
The following SQL-statements were executed during "Generate output" function.
Every single result is written right to the "-->>>" string.
These statements are only for preview and may not be executed in another SQL-Query tool!

The database was connected using the following connection-string:
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Program Files\Altova\MAPFORCE2004\MapForceExamples\Tut
*/

SELECT IIF(MAX([Altova].[PrimaryKey]) IS NULL,0,MAX([Altova].[PrimaryKey]))+1 AS [PrimaryKey] FROM [Altova]
-->>> OK. One or more rows.

INSERT INTO [Altova] ([Name],[PrimaryKey])
    VALUES ('Microtech OrgChart',%PrimaryKey%)
-->>> OK. 1 row(s).
```

Hotkeys for the Output window (keyboard and numeric key pad):

CTRL and "+"	zoom in on the text
CTRL and "-"	zoom out of the text
CTRL and "0"	resets the zoom factor to standard

CTRL and mouse wheel forward / backward achieve the same zoom in/out effect.

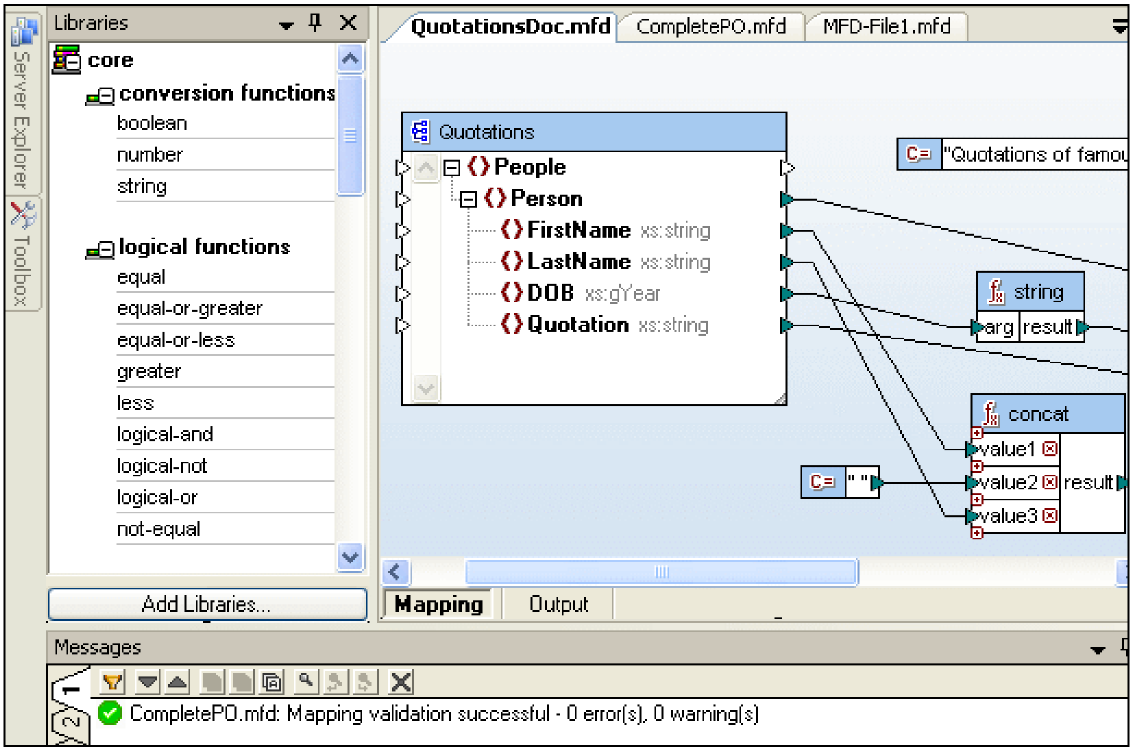
Chapter 19

MapForce plug-in for MS Visual Studio .NET

19 MapForce plug-in for MS Visual Studio .NET

You can integrate your version of MapForce2006 into the Microsoft Visual Studio .NET IDE versions 2002, 2003 and 2005. This unifies the best of both worlds, integrating advanced mapping capabilities with the advanced development environment of Visual Studio .NET. To do this, you need to do the following:

- Install Microsoft Visual Studio .NET
- Install MapForce (Enterprise or Professional Edition)
- Download and run the MapForce Visual Studio .NET Edition integration for Microsoft Visual Studio .NET package. This package is available on the MapForce (Enterprise and Professional Editions) download page at www.altova.com. (**Please note:** You must use the integration package corresponding to your MapForce edition (Enterprise or Professional).



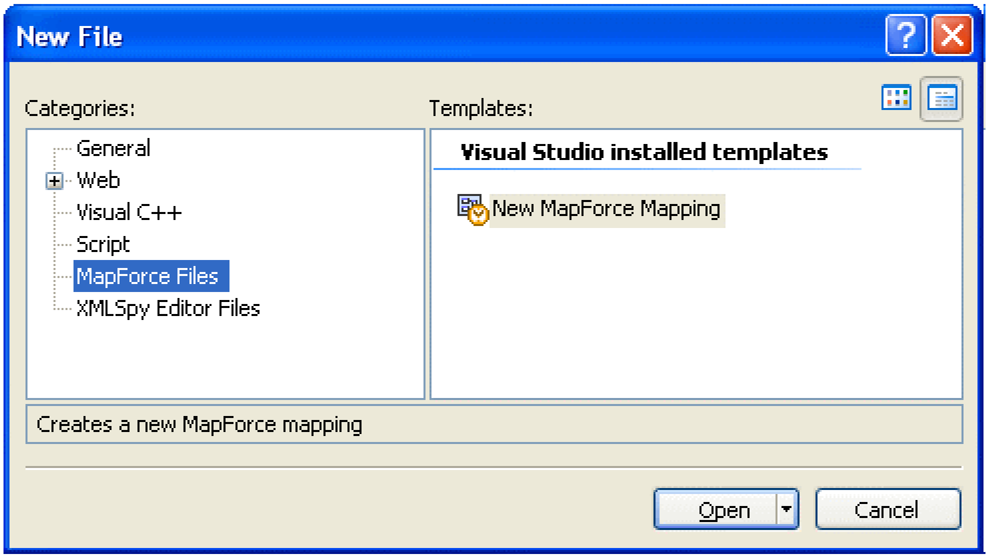
CompletePO.mfd: Mapping validation successful - 0 error(s), 0 warning(s)

Once the integration package has been installed, you will be able to use MapForce in the Visual Studio .NET environment.

19.1 Opening MapForce files in MS VS .NET

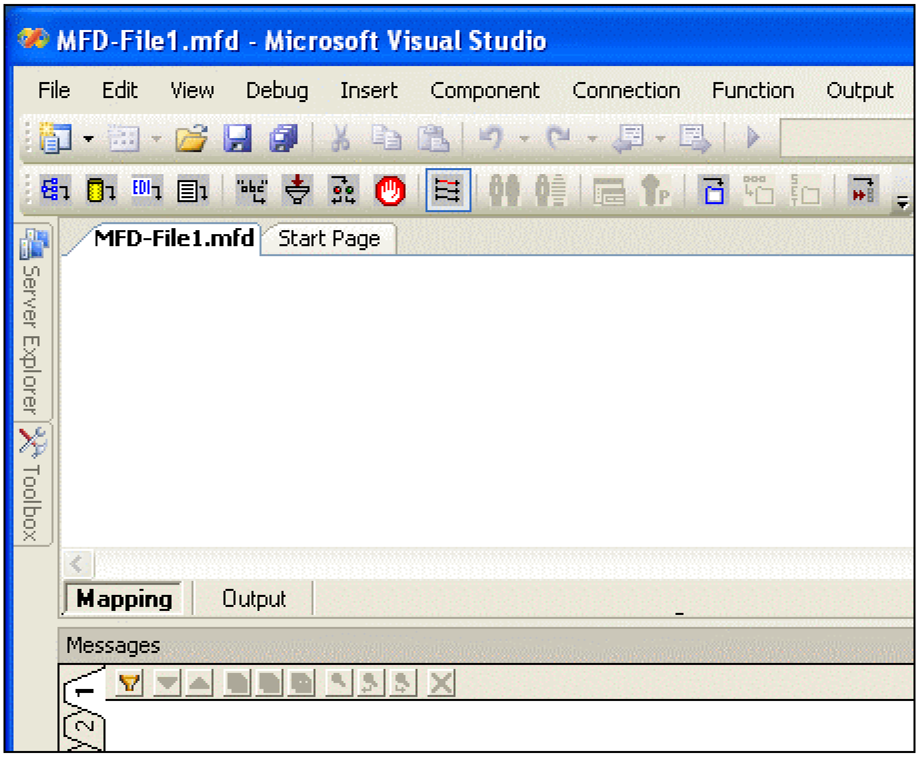
To open a new MapForce mapping file:

1. Select the menu option **File | New**.
2. Click the MapForce Files entry in Categories.



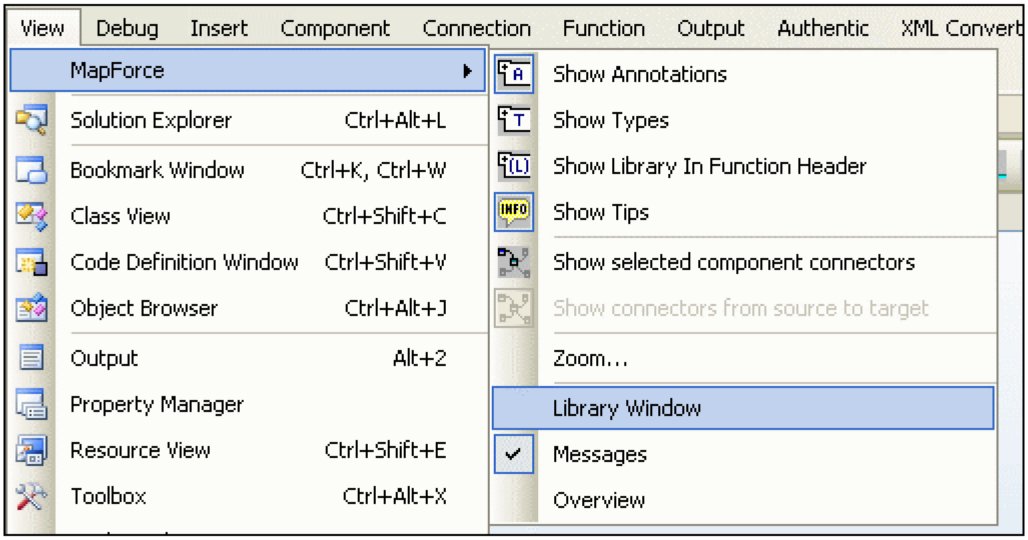
Creates a new MapForce mapping

3. Double click the "New MapForce Mapping" item in the Templates window.
   An empty mapping file is opened.



To enable the Libraries window:

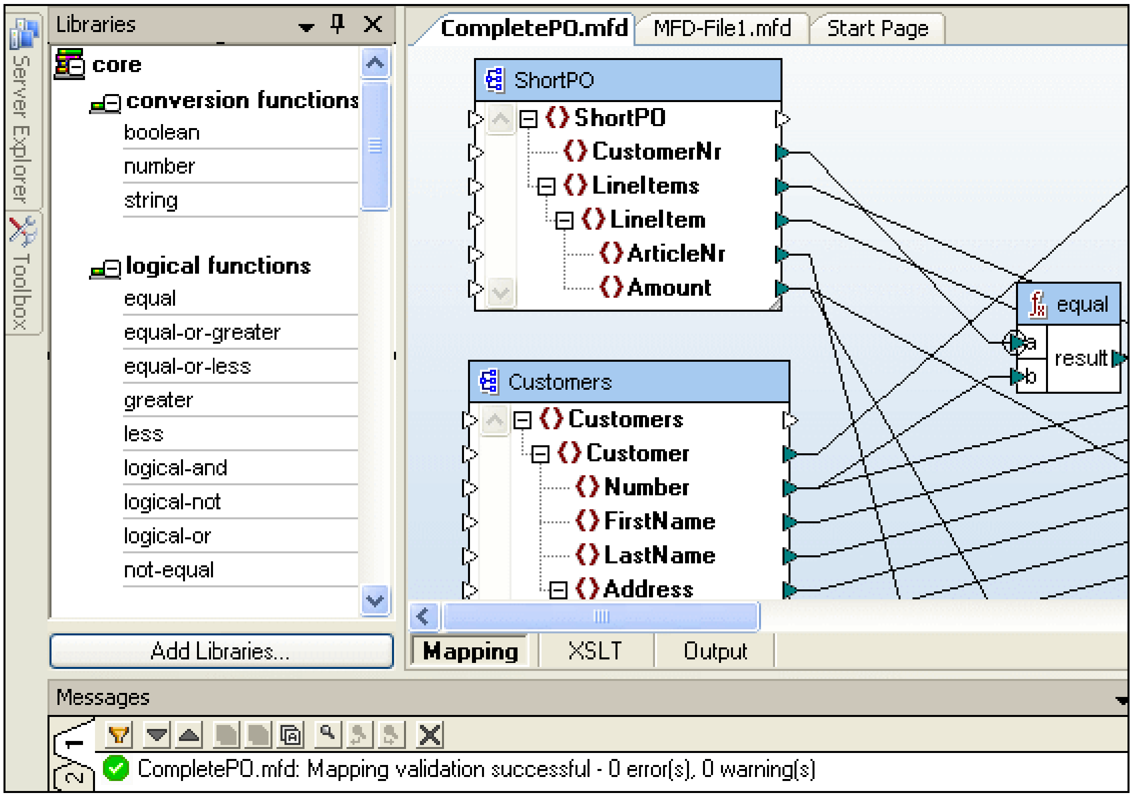
1. Select the menu item **View | MapForce | Library Window**.


View
Debug
Insert
Component
Connection
Function
Output
Authentic
XML Convert
MapForce
Solution Explorer Ctrl+Alt+L
Bookmark Window Ctrl+K, Ctrl+W
Class View Ctrl+Shift+C
Code Definition Window Ctrl+Shift+V
Object Browser Ctrl+Alt+J
Output Alt+2
Property Manager
Resource View Ctrl+Shift+E
Toolbox Ctrl+Alt+X
Show Annotations
Show Types
Show Library In Function Header
Show Tips
Show selected component connectors
Show connectors from source to target
Zoom...
Library Window
Messages
Overview

2. Dock the floating window at the position you want to use it e.g. left border.

To open an supplied sample file:

1. Select the menu option **File | Open,** navigate to the ...\MapForceExamples folder and open a MapForce file.
CompletePO.mfd is shown in the screenshot below.



CompletePO.mfd: Mapping validation successful - 0 error(s), 0 warning(s)

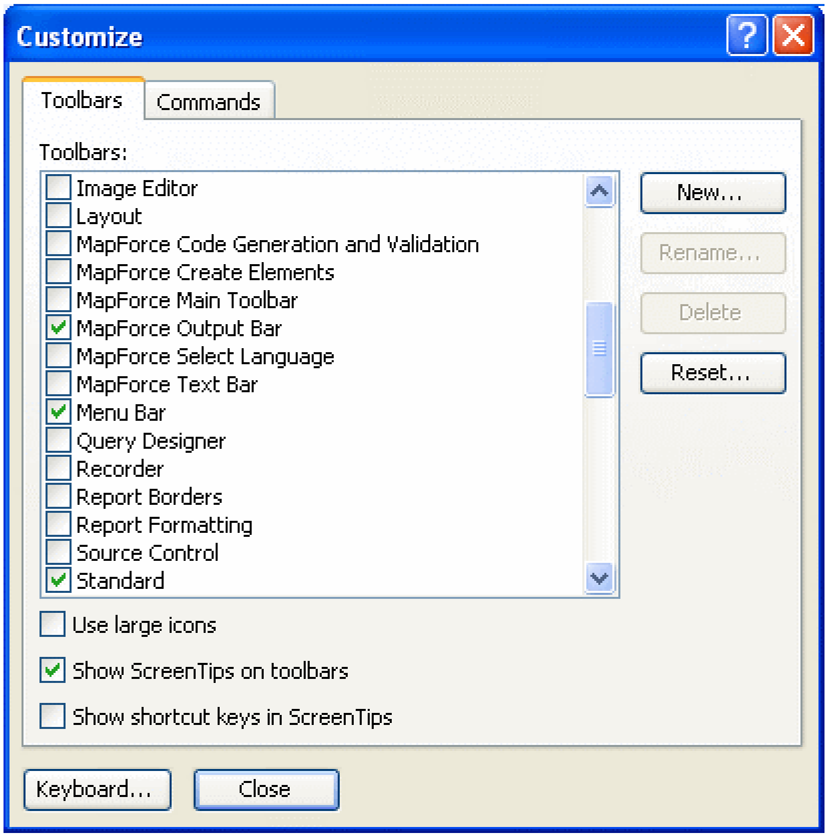
19.2 Differences between .NET and standalone versions

The Enterprise and Professional MapForce plug-ins are integrated into all versions of MS Visual Studio .NET in the same way. Please note there is no support for MapForce projects in the Visual Studio .NET version.

Changed functionality in the Visual Studio .NET editions:

Menu **Edit,** Undo and Redo
The Undo and Redo commands affect all actions (copy, paste, etc.) made in the development environment, including all actions in MapForce.

Menu **Tools | Customize | Toolbar**, **Commands**.



Toolbars:

☐ Image Editor
☐ Layout
☐ MapForce Code Generation and Validation
☐ MapForce Create Elements
☐ MapForce Main Toolbar
☑ MapForce Output Bar
☐ MapForce Select Language
☐ MapForce Text Bar
☑ Menu Bar
☐ Query Designer
☐ Recorder
☐ Report Borders
☐ Report Formatting
☐ Source Control
☑ Standard

☐ Use large icons

☑ Show ScreenTips on toolbars

☐ Show shortcut keys in ScreenTips

These tabs contain both Visual Studio .NET and MapForce commands.

Menu **View**
The View menu contains the submenu MapForce, which allows you to enable or disable the MapForce tool panes. It also gives access to MapForce view settings.

Menu **Help**
The **Help** menu contains the submenu MapForce Help, which is where you can open the MapForce help. It also contains links to the Altova Support center, Component download area, etc.

Unsupported features of the .NET edition of MapForce

Both the **Project** pane and **Project** menu are not available in these editions. This means that MapForce projects, as well as WSDL projects, cannot be opened in these editions.

Chapter 20

MapForce plug-in for Eclipse

20 MapForce plug-in for Eclipse

Eclipse 3.x is an open source framework that integrates different types of applications delivered in form of plugins. MapForce for the Eclipse Platform, is an Eclipse Plug-in that allows you to access the functionality of a previously installed MapForce Edition from within the Eclipse 3.0 and 3.1 Platform.

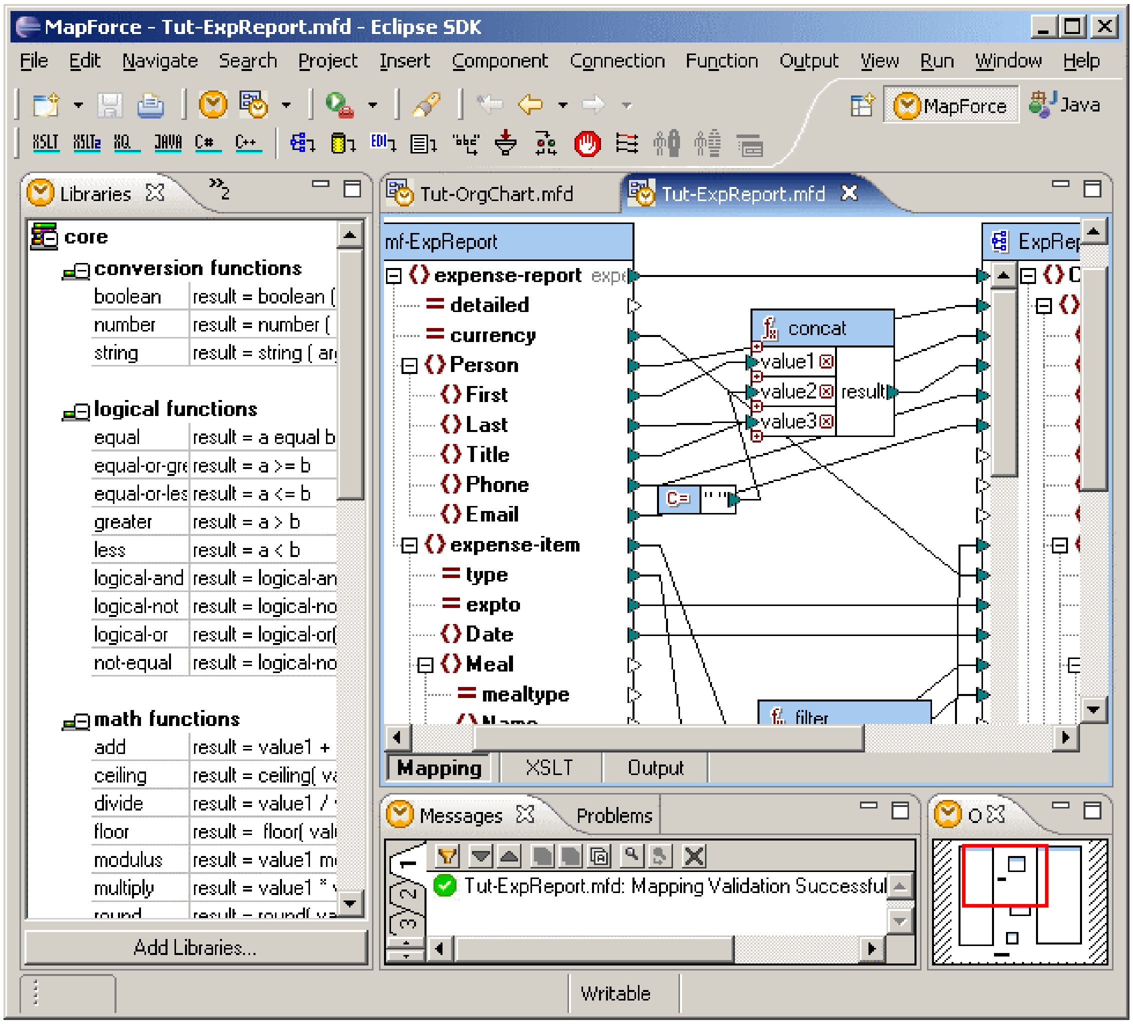
Installation Requirements

To successfully install the MapForce Plug-in for Eclipse 3.0, or 3.1, you need the following:

- The specific MapForce Edition you intend to use: Enterprise, Professional, or Standard
- The Eclipse 3.x package, as well as
- The appropriate Java Runtime Edition

The MapForce Plug-in for Eclipse supplies the following functionality:

- A fully-featured visual data mapping tool for advanced data integration projects.
- Code generation capability in the Edition specific programming languages.
- MapForce user help under the menu item **Help | MapForce| Table of Contents.**



core

conversion functions

boolean	result = boolean (
number	result = number (
string	result = string (ar

logical functions

equal	result = a equal b
equal-or-gre	result = a >= b
equal-or-les	result = a <= b
greater	result = a > b
less	result = a < b
logical-and	result = logical-an
logical-not	result = logical-no
logical-or	result = logical-or(
not-equal	result = logical-no

math functions

add	result = value1 +
ceiling	result = ceiling(va
divide	result = value1 /
floor	result = floor(valu
modulus	result = value1 m
multiply	result = value1 *
round	result = round(va

Tut-ExpReport.mfd: Mapping Validation Successful

20.1 Installing MapForce plugin

Installing the MapForce Plug-in for Eclipse:
To install the MapForce Plug-in:

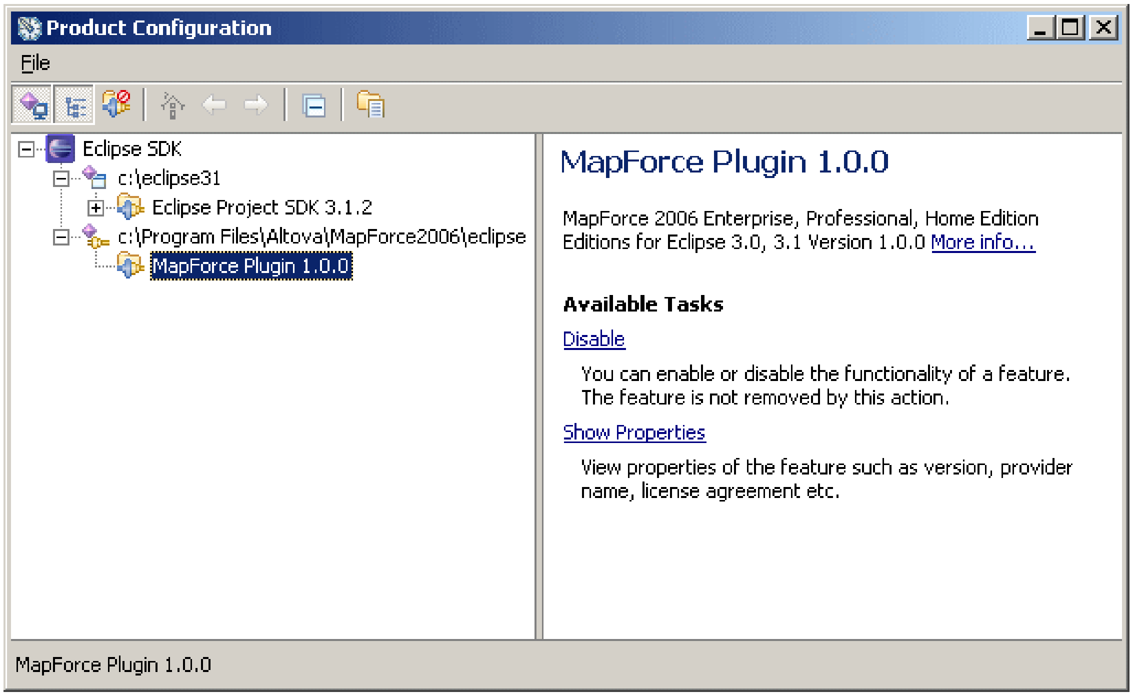
- Download and install the MapForce edition you intend to use from the Download section of the Altova.com website, i.e. Enterprise, Professional or Standard edition.
- Download and install the MapForce Plug-in for Eclipse from the Download section of the Altova.com website. You will be prompted for the installation folder of Eclipse during the installation process.

Configuring an eclipse installation to use a **previously** installed MapForce plug-in:

1. Start Eclipse and select the menu option **Help | Software Updates | Manage Configuration.**
2. Select the menu option **File | Add an extension location** and browse to the installation folder of your MapForce Eclipse plug-in e.g. C:\Program Files\Altova\MapForce2006 \eclipse.

   Follow the instructions to access a previously installed plug-in.

Clicking a plug-in or folder icon, displays various installation options in the right-hand pane.



Product Configuration
File
Eclipse SDK
c:\eclipse31
Eclipse Project SDK 3.1.2
c:\Program Files\Altova\MapForce2006\eclipse
MapForce Plugin 1.0.0

MapForce Plugin 1.0.0

MapForce 2006 Enterprise, Professional, Home Edition Editions for Eclipse 3.0, 3.1 Version 1.0.0 More info...

Available Tasks

Disable

You can enable or disable the functionality of a feature. The feature is not removed by this action.

Show Properties

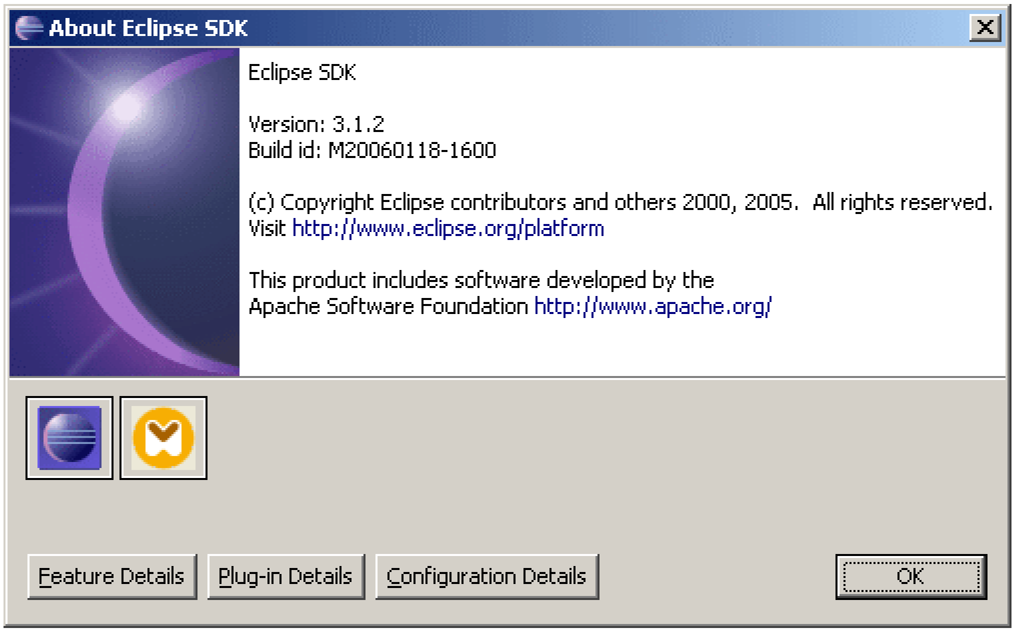
View properties of the feature such as version, provider name, license agreement etc.

MapForce Plugin 1.0.0

1. Click the "Show Properties" link displays the specific plug-in information: Copyright, General information etc.

To check the currently installed version:

1. Select the menu option **Help | About Eclipse SDK.**



Eclipse SDK

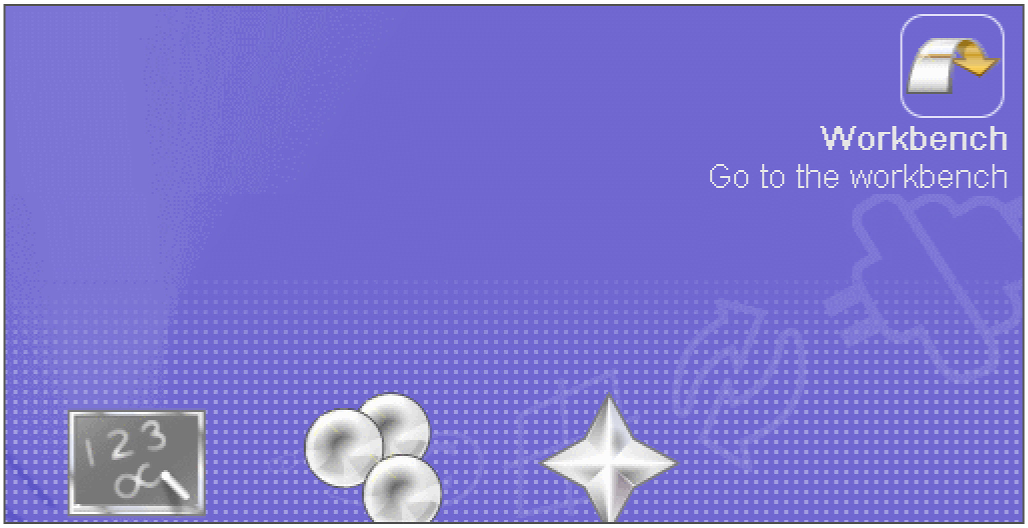
Version: 3.1.2
Build id: M20060118-1600

(c) Copyright Eclipse contributors and others 2000, 2005. All rights reserved.
Visit http://www.eclipse.org/platform

This product includes software developed by the
Apache Software Foundation http://www.apache.org/

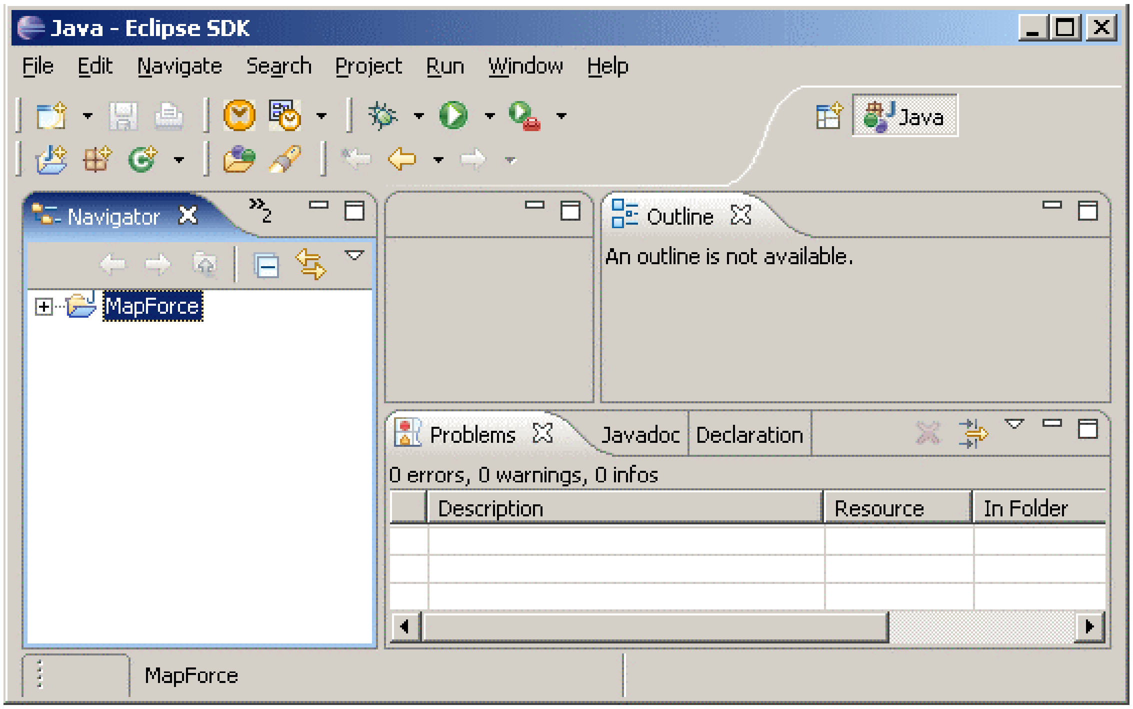
2. Click the MapForce icon, to view the version specifics.

20.2 Starting Eclipse and using MapForce plugin



Place the cursor over the arrow symbol, and click when the "Go to the workbench" text appears. This opens an empty MapForce window in Eclipse.

Starting Eclipse and Using MapForce Plug-in:
Having used the MapForce for Eclipse installer, you are presented with an empty Eclipse environment.



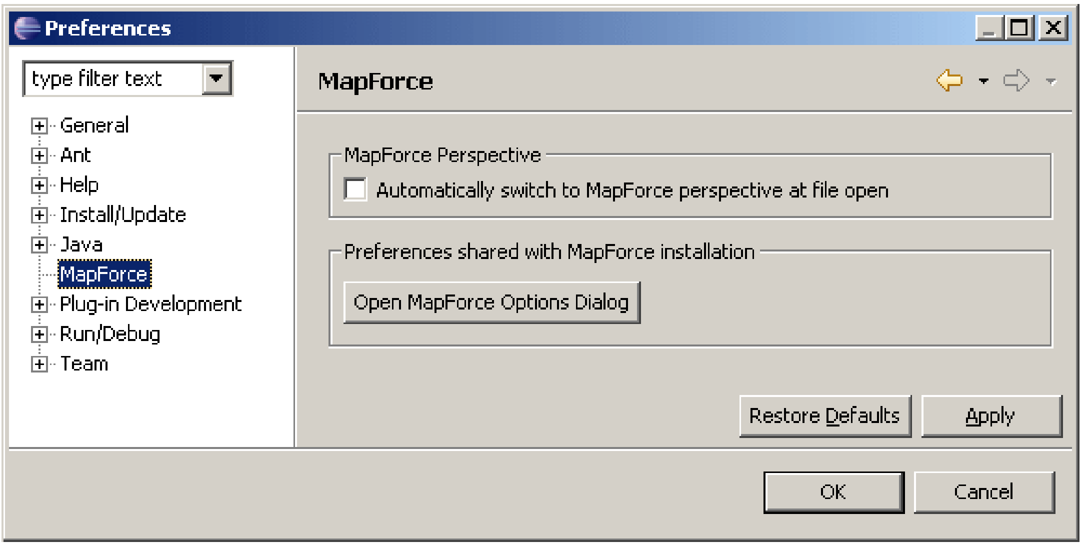
An outline is not available.

0 errors, 0 warnings, 0 infos

Description	Resource	In Folder

MapForce properties:

1. Select the menu option **Window | Preferences,** and click the MapForce entry.
2. Activate the available check box, to switch to the MapForce perspective when opening a file.



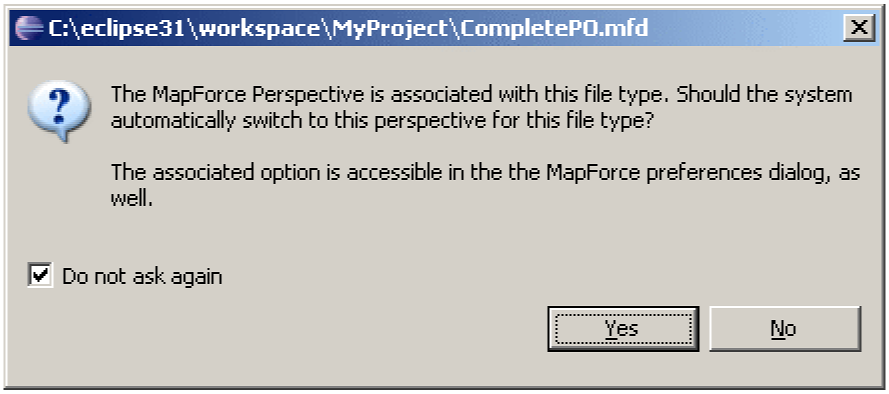
MapForce Perspective

☐ Automatically switch to MapForce perspective at file open

Preferences shared with MapForce installation

Clicking the "Open MapForce Options Dialog" button, opens the Options dialog which allows you to define the specific MapForce settings, i.e. Libraries, Code generation settings etc.

Double clicking a MapForce mapping file (*.mfd) initially opens a message box stating that a MapForce perspective is associated with this type of file, and prompts if you want Eclipse to automatically switch to the MapForce perspective in the future. These settings can be changed later through the **Window | Preferences | MapForce | MapForce Perspective** option.



C:\eclipse31\workspace\MyProject\CompletePO.mfd

The MapForce Perspective is associated with this file type. Should the system automatically switch to this perspective for this file type?

The associated option is accessible in the the MapForce preferences dialog, as well.

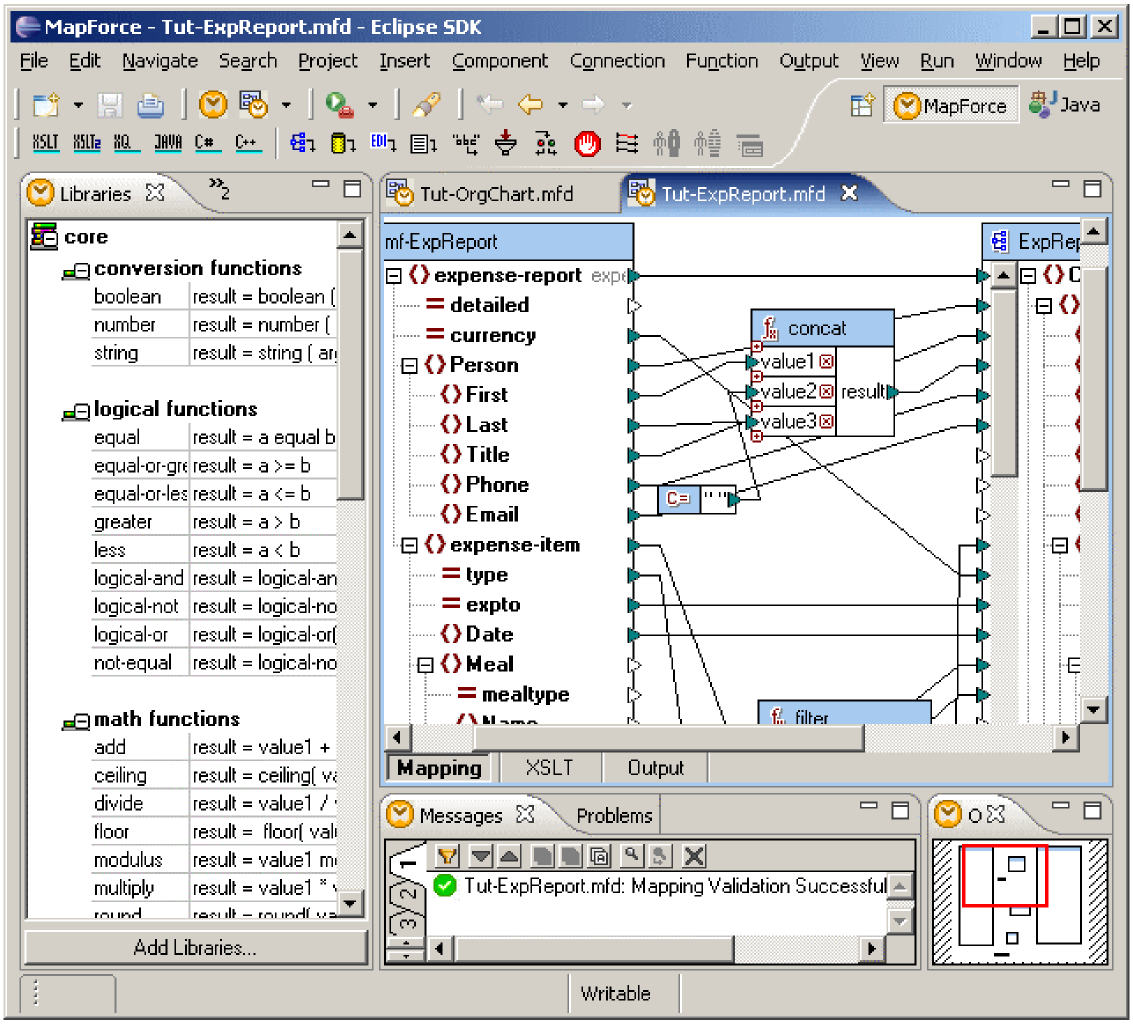
☑ Do not ask again

20.3 MapForce / Editor, View and Perspectives

The MapForce perspective can be automatically set if you activate the "Automatically switch to MapForce perspective at file open" in the **Window | Preferences** dialog box. You can also use the option described below to enable the perspective.

To enable the MapForce perspective in Eclipse:

- Select the menu option **Window | Open perspective | Other | MapForce.**



Tut-ExpReport.mfd: Mapping Validation Successful

The individual MapForce tabs are now visible in the Eclipse Environment:

- **Libraries** tab at left, allows you to select predefined or user-defined functions.
- **Messages** tab displays validation messages, errors and warnings
- **Overview** tab displays an iconized view of the mapping file.

The editor pane is where you design your mappings and preview their output, and consists of the following tabs:

Mapping, which displays the graphical mapping design.

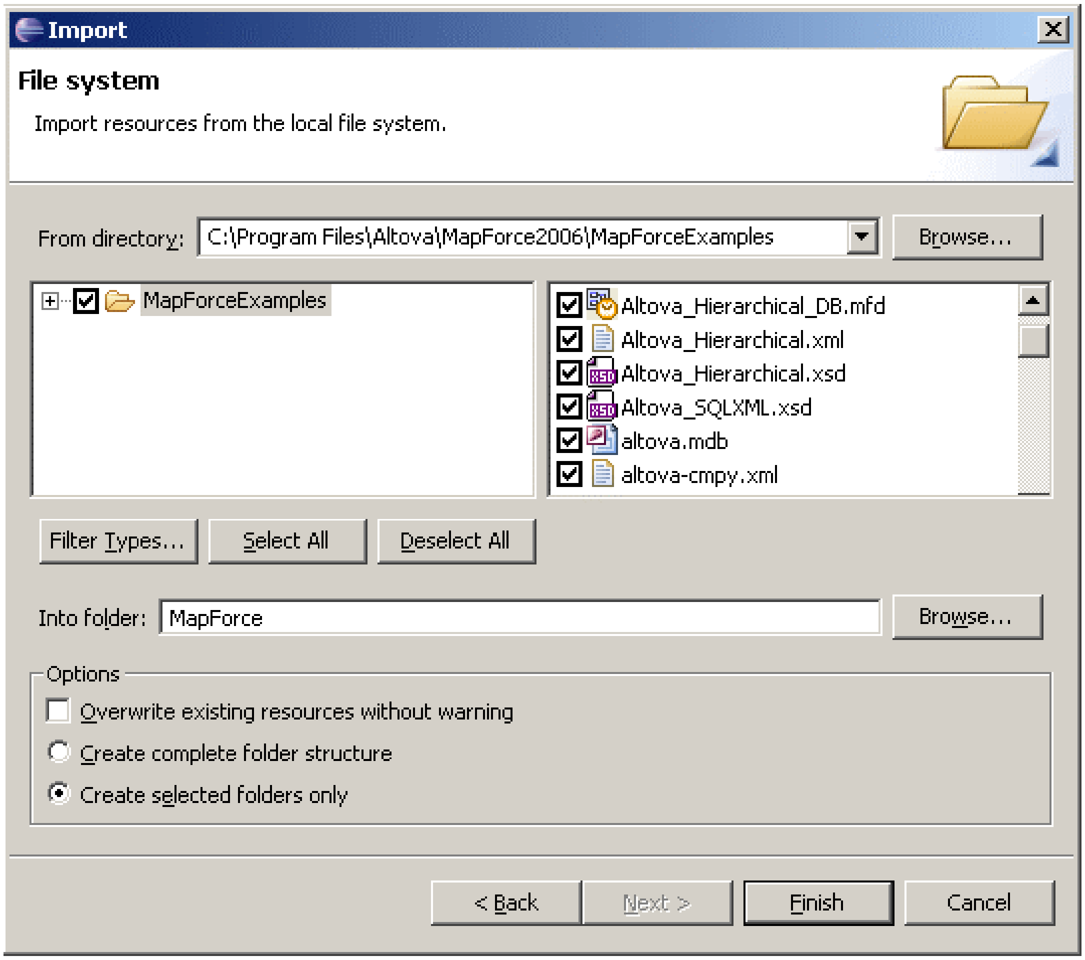
XSLT, which displays the generated XSLT code. The name of this tab reflects the programming language you have selected under Output | XSLT 1.0, Java, C# etc.

Output, which displays the Mapping output, in this case the XML data.

20.4 Importing MapForce examples folder into Navigator

To Import the MapForce Examples folder into the Navigator:

1. Right-click in the **Navigator** tab and click **Import**.
2. Select "File system", then click **Next**.
3. Click the **Browse** button to the right of the "From directory:" text box, and select the MapForceExamples directory in your MapForce folder.
4. Activate the **MapForceExamples** check box.
   This activates all files in the various subdirectories in the window at right.



Import

File system

Import resources from the local file system.

From directory: C:\Program Files\Altova\MapForce2006\MapForceExamples

☑ MapForceExamples

☑ Altova_Hierarchical_DB.mfd
☑ Altova_Hierarchical.xml
☑ Altova_Hierarchical.xsd
☑ Altova_SQLXML.xsd
☑ altova.mdb
☑ altova-cmpy.xml

Into folder: MapForce

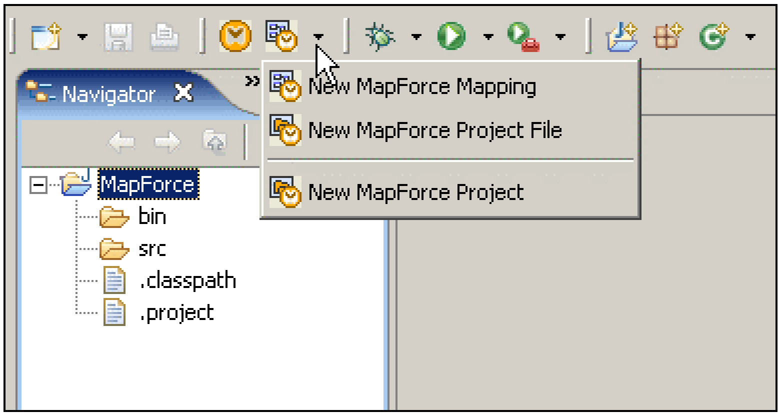
Options
□ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

5. If not automatically supplied, click the **Browse** button, next to the "Into folder:" text box, to select the target folder, then click **Finish**.
   The selected folder structure and files will be copied into the Eclipse workspace.
6. Double-click a file in Navigator to open it.

20.5 Creating new MapForce files (mapping and project file)

To create a new MapForce mapping or project files:

- Click the New MapForce... combo box and select the required option.



- New MapForce mapping, creates a single mapping file.
- New MapForce Project File, creates a MapForce project that can combine multiple mappings into one code-generation unit. You must select this option when you are creating webservices.
- New MapForce Project, creates a new MapForce/Eclipse project, adding the folder to the Navigator window. MapForce/Eclipse projects are Eclipse projects with a MapForce builder assigned to them. See Using MapForce Eclipse projects for automatic build for details.

20.6 MapForce code generation

Build Integration
MapForce mappings can be contained in any eclipse project. Generation of mapping code can be triggered manually by selecting one of the **'Generate Code...**' menu entries for the mapping or MapForce project file. Full integration into the Eclipse auto-build process is achieved by assigning the MapForce builder to an Eclipse project.

For manual code generation see Build mapping code manually

For automatic generation of mapping code please see Using MapForce Eclipse projects for automatic build and Adding MapForce nature to existing Eclipse Project.

20.6.1 Build mapping code manually

To manually build mapping code for a single mapping:

1. Open, or select the mapping, and select **File | Generate Code in**, or **File | Generate Code in Selected Language.**
   You are prompted for a target folder for the generated code.
2. Select the folder and click OK to start code generation.
   Any errors or warnings are displayed in the MapForce Messages tab.

To manually build mapping code for multiple mappings combined into a MapForce **project:**

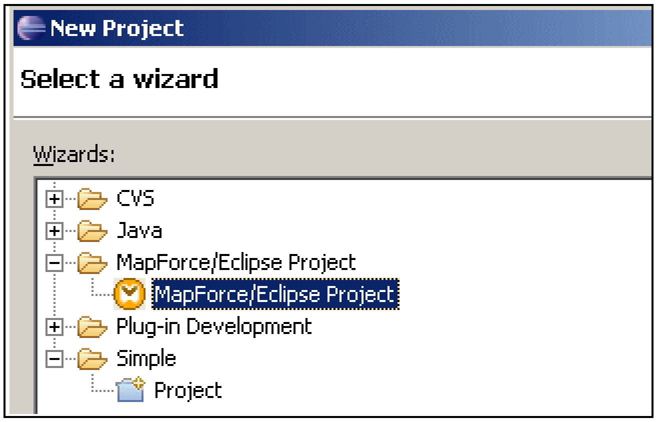
1. Open, or select the MapForce project file.
2. Select the root node or any other node in the project document.
3. Select **Generate Code**, or **Generate Code in** from the right mouse-button menu**.**
   The target folder for the generated code is determined by the properties of the selected node or properties of its parents.
4. Any errors or warnings are displayed in the MapForce Messages tab.

20.6.2 Using MapForce Eclipse projects for automatic build

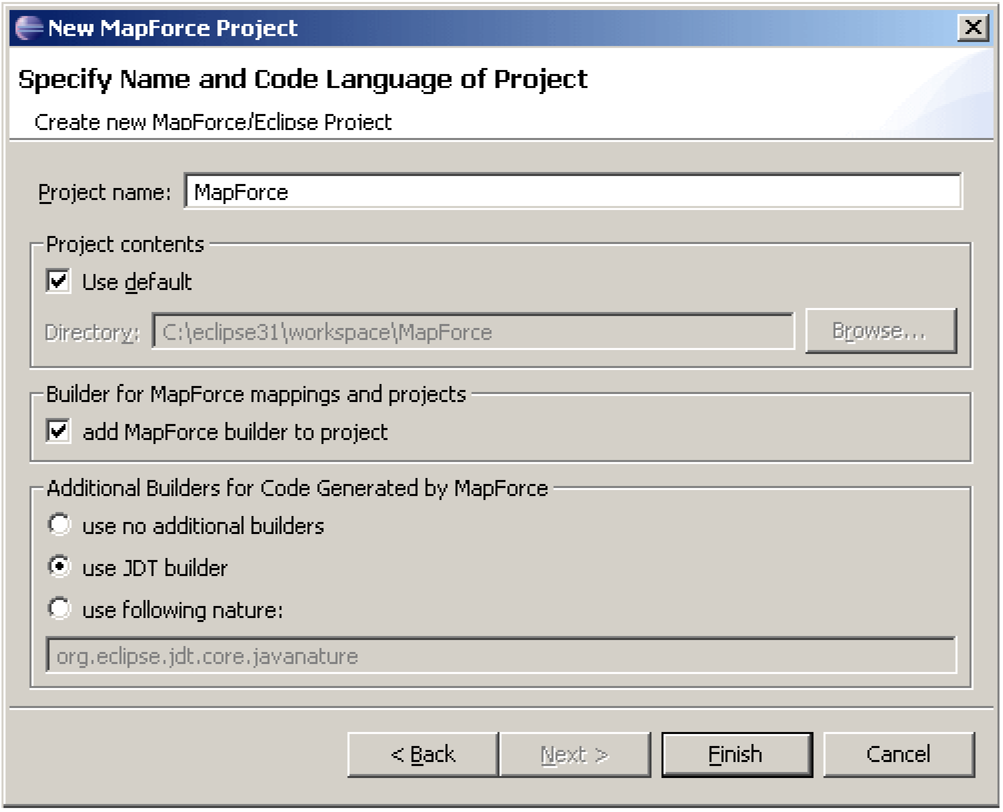
The MapForce plug-in has a built-in project builder. This builder can be identified by the project nature ID `"com.altova.mapforceeclipseplugin.MapForceNature"`. MapForce Eclipse projects have this nature automatically assigned. To use the MapForce project builder in other Eclipse projects see "Adding MapForce nature to existing Eclipse Project" for more information.

To create a new MapForce Eclipse Project:

1. Click the Navigator tab to make it active.
2. Right-click in the Navigator window, and select **New | Project.**
3. Expand the MapForce/Eclipse Project entry and select MapForce/Eclipse, then click Next.



4. Enter the project name (e.g. MapForce) and change any of the other project settings to suit your environment, then click Finish. Note the default setting in the "Additional Builders..." group, use JTD builder.



New MapForce Project

Specify Name and Code Language of Project

Create new MapForce/Eclipse Project

Project name: MapForce

Project contents

☑ Use default

Directory: C:\eclipse31\workspace\MapForce Browse...

Builder for MapForce mappings and projects

☑ add MapForce builder to project

Additional Builders for Code Generated by MapForce

○ use no additional builders

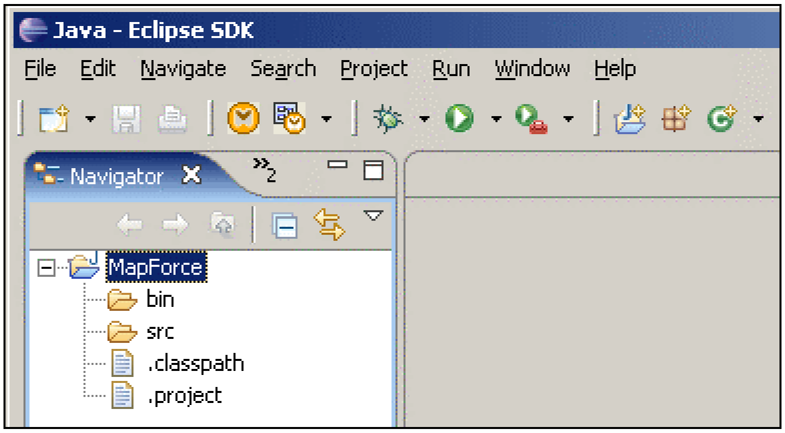
● use JDT builder

○ use following nature:

org.eclipse.jdt.core.javanature

< Back Next > Finish Cancel

An Eclipse project folder and optionally some more folders and files inside this folder have been created.



You can now create MapForce mappings and MapForce project files inside this Eclipse project, or copy existing ones into it. Whenever a mapping or MapForce project file changes, the corresponding mapping code will be generated automatically. Code generation errors and warnings will be shown in the MapForce view called **Messages** and added to the **Problems** view of Eclipse.

A MapForce Eclipse project is an Eclipse project with the MapForce nature assigned to it, and therefore uses the MapForce builder.

If one or more MapForce project files are present in the Eclipse project, the code generation language and output target folders are determined by the settings in these files.

If a MapForce project file is not present in the Eclipse project:

But the Eclipse project has been assigned the JDT nature:

- Then, the mapping code generation defaults to Java language, and the project's Java source code directory is used as the mapping code output directory.

  Saving a mapping automatically generates the mapping code in Java and compilation of the Java code. Use the Java debug or run command, to test the resulting mapping application.

But the project has **not** been assigned the JDT nature:

- Then the output target folder is the project folder, and the code generation language defaults to the current setting in the MapForce Options.

To activate the Automatic Build process:

1. Make sure that the menu option **Project | Build automatically** is checked.

To temporarily deactivate automatic building of MapForce mapping code:

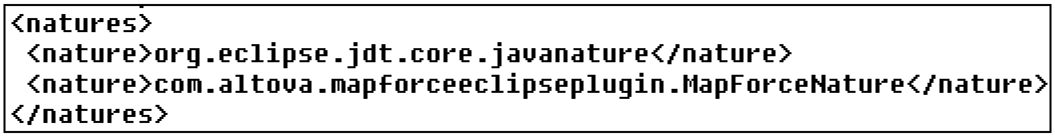
This is only available to Eclipse projects that have added the MapForce nature.

1. Right click the Eclipse project**,** in the Navigator pane.
2. Select **Properties** from the context menu.
3. Click the "Builders" entry in the left pane of the project properties dialog.
4. Un-check the **MapForce builder** check box in the right pane.
   Modifications to any mapping files or MapForce project files in this Eclipse project, will now no longer trigger automatic generation of mapping code.

20.6.3 Adding MapForce nature to existing Eclipse Project

Applying the MapForce Nature to Existing Projects:
Add the following text to the **natures** section of the **.project** file in the Eclipse project (e.g. in the c:\eclipse31\workspace\MapForce\ folder):

<nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>



```
<natures>
 <nature>org.eclipse.jdt.core.javanature</nature>
 <nature>com.altova.mapforceeclipseplugin.MapForceNature</nature>
</natures>
```

Any MapForce project files and mappings contained in this project will now participate in the automatic build process. For MapForce specific details see Using MapForce Eclipse projects for automatic build.

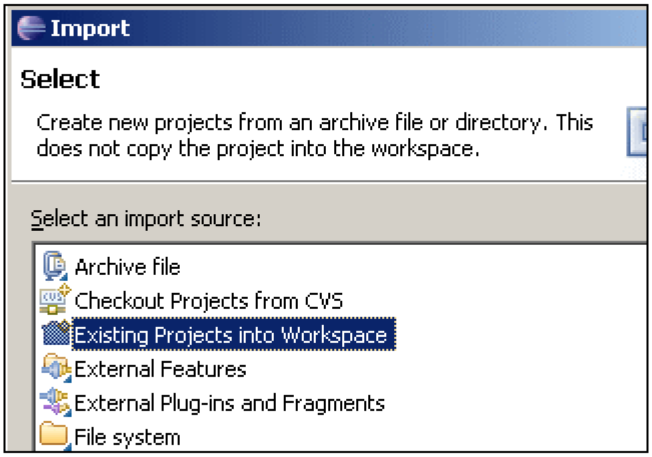
20.7 Extending MapForce plug-in

MapForce plug-in provides an Eclipse extension point with the ID "com.altova.mapforceeclipseplugin.MapForceAPI". You can use this extension point to adapt, or extend the functionality of the MapForce plug-in. The extension point gives you access to the COM-Interface of the MapForce control and the MapForceAPI.

Your MapForce Eclipse installation package contains a simple example of a plug-in that uses this extension point. It checks for any file open events of any new MapForce mappings, and sets the zoom level of the mapping view to 70%.

Installing the Sample extension plug-in:
MapForce plug-in requires the JDT (Java Development Tools) plug-in to be installed.

1. Start Eclipse.
2. Right click in **Navigator** or PackageExplorer, and select the menu item **Import**.
3. Select "Existing projects into Workspace, and click Next.



Import

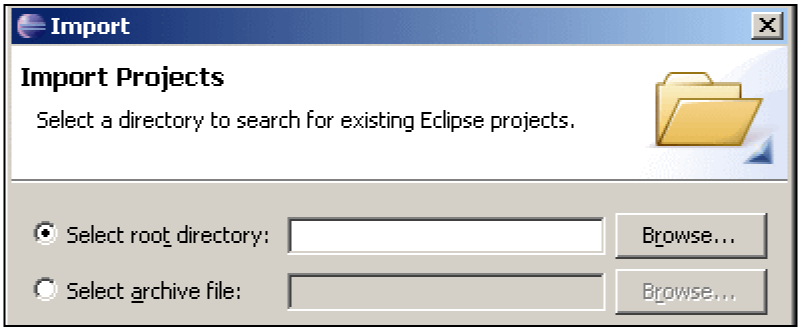
Select

Create new projects from an archive file or directory. This does not copy the project into the workspace.

Select an import source:

Archive file
Checkout Projects from CVS
Existing Projects into Workspace
External Features
External Plug-ins and Fragments
File system

4. Click the **Browse**... button next to the "'Select root directory" field and choose the sample project directory e.g. **C:\Program Files\Altova\MapForce2006\eclipse\workspace\MapForceExtension**).



Import

Import Projects

Select a directory to search for existing Eclipse projects.

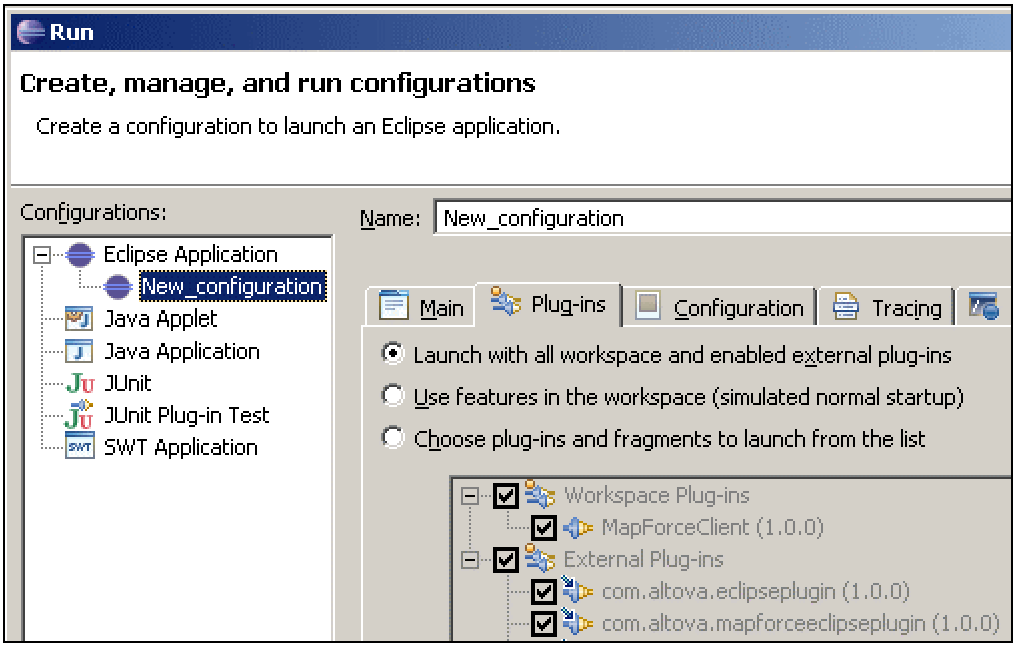
5. Click Finish.
   A new project named "MapForceExtension" has been created in your workspace.

Accessing javadoc for the extension point of MapForce plug-in:

1. Open the **index.html** in the docs folder of the plugin installation e.g. c:\Program Files\Altova\MapForce2006\eclipse\plugins\com.altova.mapforceeclipseplugin_1.0.0\docs\

Running the Sample extension plug-in:

1. Switch to the Java perspective.
2. Select the menu option **Run | Run...**
3. Select Eclipse Application and click **New_configuration**.



Run

Create, manage, and run configurations

Create a configuration to launch an Eclipse application.

Configurations:

- Eclipse Application
  - New_configuration
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- SWT Application

Name: New_configuration

● Launch with all workspace and enabled external plug-ins

○ Use features in the workspace (simulated normal startup)

○ Choose plug-ins and fragments to launch from the list

- ☑ Workspace Plug-ins
  - ☑ MapForceClient (1.0.0)
- ☑ External Plug-ins
  - ☑ com.altova.eclipseplugin (1.0.0)
  - ☑ com.altova.mapforceeclipseplugin (1.0.0)

4. Check that the project MapForceClient is selected in the 'Plug-ins' tab.
5. Click the Run button.
   A new Eclipse Workbench opens.
6. Open any MapForce mapping in the new Workbench. It will now open with a zoom level of 70%.

Chapter 21

MapForce Reference

21 MapForce Reference

The following section lists all the menus and menu options in MapForce, and supplies a short description of each.

21.1 File

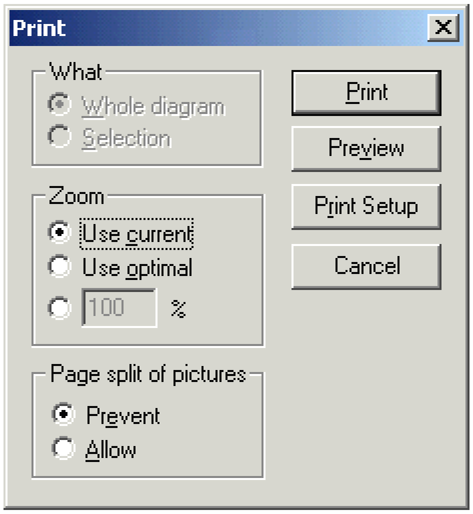
New
Clears the Mapping tab, if a previous mapping exists, and creates a new mapping document.

Open
Opens previously defined mapping (*.mfd) files.

Save
Saves the currently active mapping using the currently active file name.

Save As
Saves the currently active mapping with a different name, or allows you to supply a new name if this is the first time you save it.

Print
Opens the Print dialog box, from where you can printout your mapping as hardcopy.



Print

What
● Whole diagram
○ Selection

Zoom
● Use current
○ Use optimal
○ 100 %

Page split of pictures
● Prevent
○ Allow

"Use current", retains the currently defined zoom factor of the mapping. "Use optimal" scales the mapping to fit the page size. You can also specify the zoom factor numerically. Please note that component scrollbars are not printed. You can also specify if you want to allow the graphics to be split over several pages or not.

Print Preview
Opens the same Print dialog box with the same settings as described above.

Print Setup
Open the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

Validate Mapping
Validating a Mapping validates:

- that all mappings (connectors) are valid
- please note, that the current release supports mixed content mapping.

Please see "Validating mappings" for more information

Generate code in selected language
Generates code in the currently selected language of your mapping. The currently selected

language is visible as a highlighted programming language icon in the title bar XSLT, XSLT 2 XQuery, Java, C#, or C++.

Generate code in | XSLT (XSLT2)
This command generates the XSLT file(s) needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box where you select the location of the XSLT file.

Note: the name of the generated XSLT file(s) is defined in the **Application Name** field of the Mapping Output dialog box. This dialog is opened by selecting **File | Mapping Settings** menu option. A notification appears when the process has been completed successfully.

Generate code in | XQuery
This command generates the XQuery file(s) needed for the transformation from the source file(s).
Selecting this option opens the Browse for Folder dialog box where you select the location of the XQuery file.

Note: the name of the generated XQuery file(s) is defined in the **Application Name** field of the Mapping Output dialog box. This dialog is opened by selecting **File | Mapping Settings** menu option. A notification appears when the process has been completed successfully.

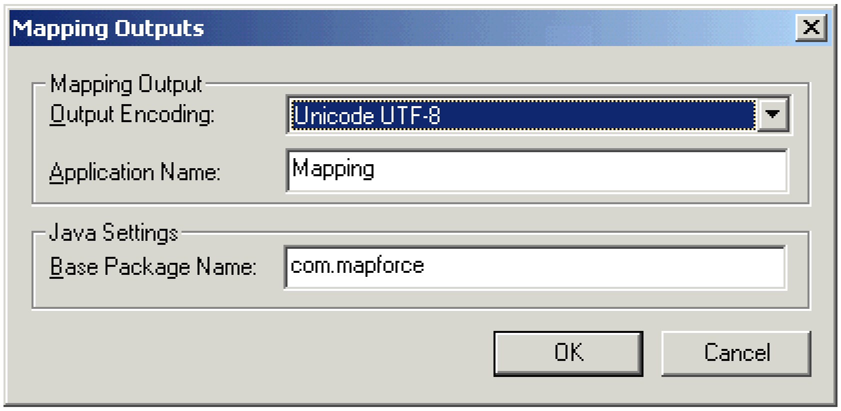
Generate code in | Java
This command generates the Java file(s) needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box, where you select the location of the Java files.

Note: the name of the generated Java file(s) are defined in the **Application Name** field of the Mapping Output dialog box. This dialog is opened by selecting **File | Mapping Settings** menu option. A notification appears when the process has been completed successfully.

Generate code in | C#
Generates the C# code in the selected directory. The file name created by the executed code, is that which appears in the **Output XML instance (for Code Generation)** field of the Component settings dialog box if the target is an XML/Schema document.

Generate code in | C++
Generates the C++ code in the selected directory. The file name created by the executed code, is that which appears in the **Output XML instance (for Code Generation)** field of the Component settings dialog box if the target is an XML/Schema document.

Mapping settings


Mapping Outputs

Mapping Output

Output Encoding: Unicode UTF-8

Application Name: Mapping

Java Settings

Base Package Name: com.mapforce

The global MapForce settings are defined here.

Mapping Output
Output encoding: defines the output encoding for the files produced by the XSLT 1.0/2.0 XQuery and Java transformation.

Application Name: defines both the XSLT1.0/2.0 file name prefix, Java, C# or C++ application name for the transformation files.

Java settings
Base Package Name: defines the base package name for the Java output.

21.2 Edit

Most of the commands in this menu, become active when you view the result of a mapping in the **Output** tab, or preview XSLT code in the XSLT tab.

Undo
MapForce has an unlimited number of "Undo" steps that you can use to retrace you mapping steps.

Redo
The redo command allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.

Find
Allows you to search for specific text in either the XSLT, XSLT2, XQuery, or Output tab.

Find Next F3
Searches for the next occurrence of the same search string.

Cut/Copy/Paste/Delete
The standard windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window.

Select all
Selects all the text/code in the XSLT, XSLT2, XQuery or Output tab.

21.3 Insert

XML Schema / File

Inserts an XML schema file into the mapping tab. You are then prompted if you want to include an XML instance file which supplies the data for the XSLT, XSLT2, XQuery, and Output previews, as well as the for the XML target schema / document.

Database

Inserts a schema component with a database as the data source. The database supplies the data for the schema component and displays it in a tree view.

Text file

Inserts a flat file document, i.e. CSV, or a fixed length text file. Both types of file be used as source and target components.

Constant

Inserts a constant which is a function component that supplies fixed data to an input icon. The data is entered into a dialog box when creating the component. There is only one output icon on a constant function. You can select the following types of data: String, Number and All other.

Filter: Nodes/Rows

Inserts a component that uses two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Boolean is true, then the value of the node/row parameter is forwarded to the on-true parameter. If the Boolean is false, then the complement value is passed on to the on-false parameter. Please see the tutorial example on how to use a filter.

IF-Else Condition

A condition is a component which allows you to pass on different sets of data depending on the outcome of a preset condition. The component header displays the text "**if-else"**.

- The first input parameter is a **bool**, which contains the data you are checking against.
- The **value-true** input parameter supplies the data to be passed on, as a result, if the condition is true.
- The **value-false** supplies the data to be passed on if the condition is false.
- The **result** parameter outputs the data supplied by the value-true/false input parameters.

Exception

The exception component allows you to interrupt a mapping process when a specific condition is met. Please see MapForce Exceptions, for more information.

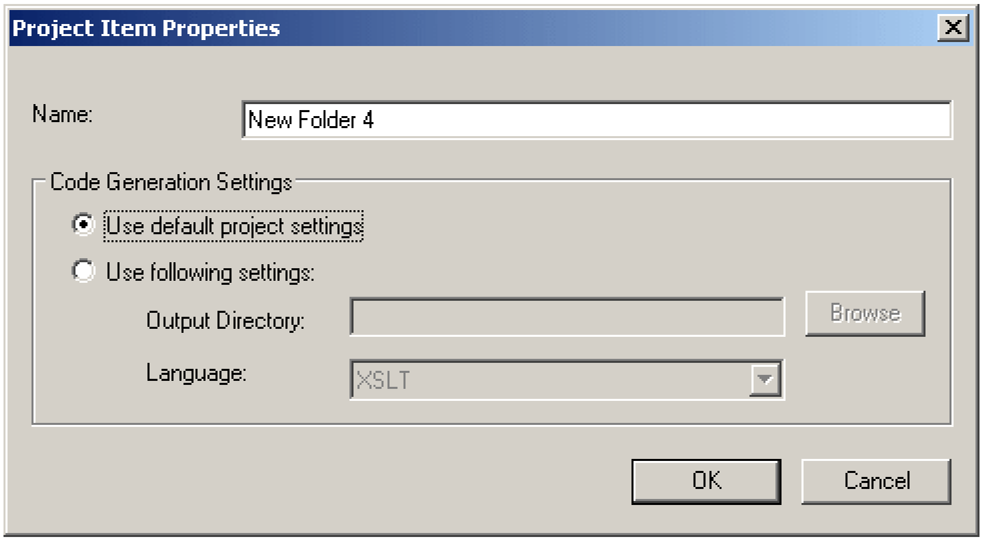
21.4 Project

MapForce supports the Multiple Document Interface and allows you to group your mappings into mapping projects. Project files have a ***.mfp** extension.

Add files to project:
Allows you to add mappings to the current project through the Open dialog box.

Add active file to project:
Adds the currently active file to the currently open project.

Create Folder:
This option adds a new folder to the current project structure, and only becomes active when this is possible.
The default project settings can be applied, or you can define your own by clicking the "Use following settings" radio button.



Project Item Properties

Name: New Folder 4

Code Generation Settings

● Use default project settings

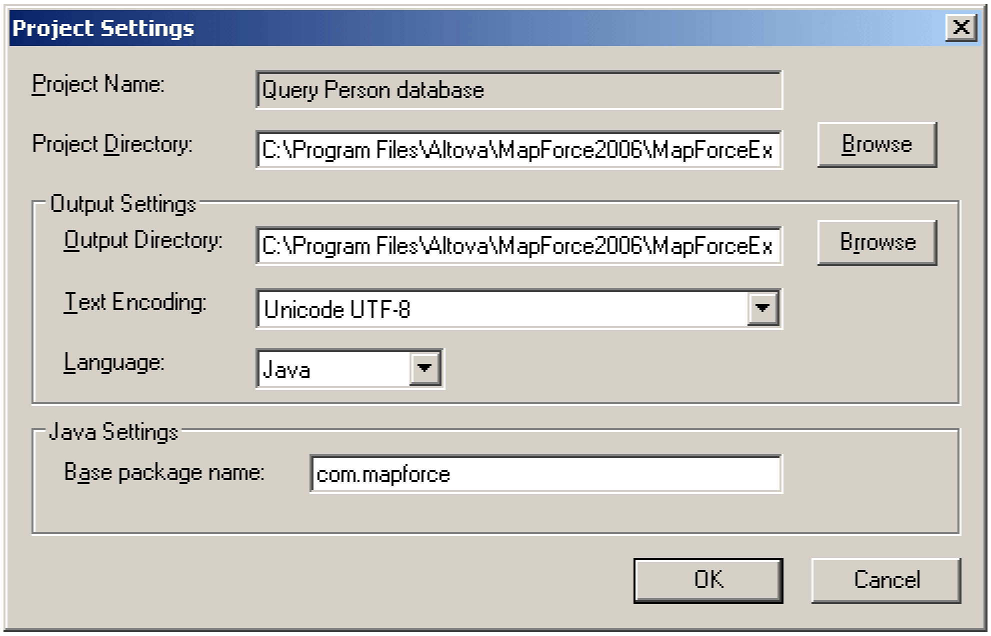
○ Use following settings:

Output Directory:

Language: XSLT

Remove item:
Removes the currently selected item from the project tree.

Project properties:
Opens the Project properties dialog box.



Project Settings

Project Name: Query Person database

Project Directory: C:\Program Files\Altova\MapForce2006\MapForceEx

Output Settings

Output Directory: C:\Program Files\Altova\MapForce2006\MapForceEx

Text Encoding: Unicode UTF-8

Language: Java

Java Settings

Base package name: com.mapforce

Generate code in default language:
Generates project code in the currently selected default language. Right click the project name in the Project window, and select **Project settings** to define the default language.

Generate code in...
Generates project code in the language you select from the flyout menu.

21.5 Component

Edit Constant

Allows you to change the entry currently defined in the Constant component. A Constant is added by clicking the **Insert Constant** icon "abc".

Align tree left

Aligns all the items along the left hand window border.

Align tree right

Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Change Root element

Allows you to change the root element of the XML instance document. Useful in the target schema window, as this limits or preselects the schema data.

Edit Schema Definition in XMLSpy

Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Duplicate input

Inserts a copy/clone of the selected item, allowing you to map multiple input data to this item. Duplicate items do not have output icons, you cannot use them as data sources. Please see the Duplicating input items section in the tutorial for an example of this.

Remove duplicate

Removes a previously defined duplicate item. Please see the Duplicating input items section in the tutorial for more information.

Database Table actions

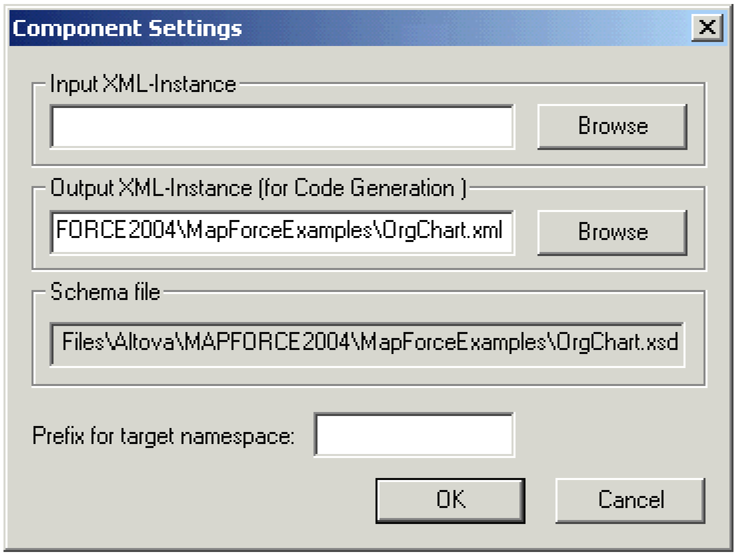
Allows you to define the table actions to be performed on the specific target database table. Table actions are: Insert, Update, and Delete, please see Mapping data to databases for more information.

Database Key settings

Allows you to define the Key settings of database fields, please see Database key settings for more information.

Component Settings

Opens a dialog box which displays the currently selected component settings. If the component is an XML-Schema file then the Component Settings dialog box is opened. If the component is a Text file, then the "Text import / export" dialog box is opened.


Component Settings

Input XML-Instance

Output XML-Instance (for Code Generation): FORCE2004\MapForceExamples\OrgChart.xml

Schema file: Files\Altova\MAPFORCE2004\MapForceExamples\OrgChart.xsd

Prefix for target namespace:

Input XML-Instance: Allows you to select, or change the XML-Instance for the currently selected schema component. This field is filled when you first insert the schema component and assign an XML-instance file.

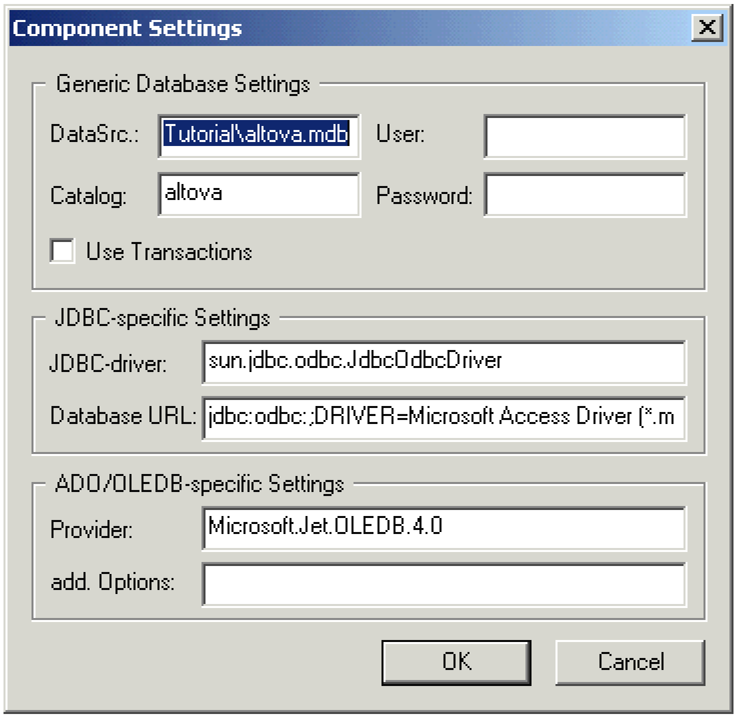
Output XML-Instance for code generation: This is file name and path where the XML target instance is placed, when generating and executing program code.

The entry from the Input XML-Instance field, is automatically copied to this field when you assign the XML-instance file. If you do not assign an XML-Instance file to the component, then this field contains the entry **schemafilenameandpath.xml.**

Schema file: Shows the file name and path of the target schema.

Prefix for target namespace: Allows you to enter a prefix for the Target Namespace if this is a schema / XML document. A Target namespace has to be defined in the target schema, for the prefix to be assigned here.

The database settings for this dialog box are only displayed if you open the component settings dialog box of a database component.



DataSrc.: Tutorial\altova.mdb
User:
Catalog: altova
Password:
☐ Use Transactions
JDBC-driver: sun.jdbc.odbc.JdbcOdbcDriver
Database URL: jdbc:odbc:;DRIVER=Microsoft Access Driver (*.m
Provider: Microsoft.Jet.OLEDB.4.0
add. Options:

Please note:
The database settings defined in this dialog box only affect the Output tab (the preview), or the generated application, they do not override the **original** database settings/selections made when **inserting** the database component i.e. (connection string, selected tables, owners, views etc.) .

Modifications made here, i.e. changing the table name in the DataSc field, only affect the SQL code in the Output tab, and the generated program code.

- opening the MFD file, accesses the **original** database component settings
- the generated application, and the pseudo SQL-code in the Output window, use the modified settings of the component.

Generic Database settings

DataSrc: displays the data source name.

Catalog: displays the name of the specific database.

User: Enter the user name needed to access the database, if required.

Password: Enter the password needed to access the database, if required

Use Transactions: Enables transaction processing when using a database as a target. A dialog box opens when an error is encountered allowing you to choose how to proceed. Transaction processing is enabled for all tables of the database component when you select this option.

JDBC -specific Settings

JDBC driver: Displays the currently active driver for the database component. The default driver is automatically entered when you define the database component. You can change the driver entered here to suit your needs. Please make sure that the syntax of the entry in the Database URL field, conforms to the specific driver you choose.

Database URL: URL of the currently selected database. Make sure that this entry conforms to the JDBC driver syntax, of the specific driver entered in the JDBC-driver field.

User: Enter the user name needed to access the database, if required.

Password: Enter the password needed to access the database, if required.

ADO/OLEDB-specific settings:

Provider: Displays the currently active provider for the database component. The provider is automatically entered when you define the database component.

add. Options: Displays additional database options.

21.6 Connection

Auto Connect Matching Children

Activates or de-activates the "Auto connect child items" function, as well as the icon in the icon bar.

Settings for Connect Matching Children

Opens the Connect Matching Children dialog box in which you define the connection settings.

Connect Matching Children

This command allows you to create multiple connectors for items of the **same name,** in both the source and target schemas. The settings you define in this dialog box are retained, and are applied when connecting two items, if the "**Auto connect child items**" icon in the title bar is active. Clicking the icon, switches between an active and inactive state. Please see the section on Connector properties for further information.

Target Driven (Standard)

Changes the connector type to Standard mapping, please see: "Source-driven / mixed content vs. standard mapping" for more information.

Copy-all

Creates connectors for all matching child items, where each of the child connectors are displayed as a subtree of the parent connector, please see "Copy-all connections" for more information.

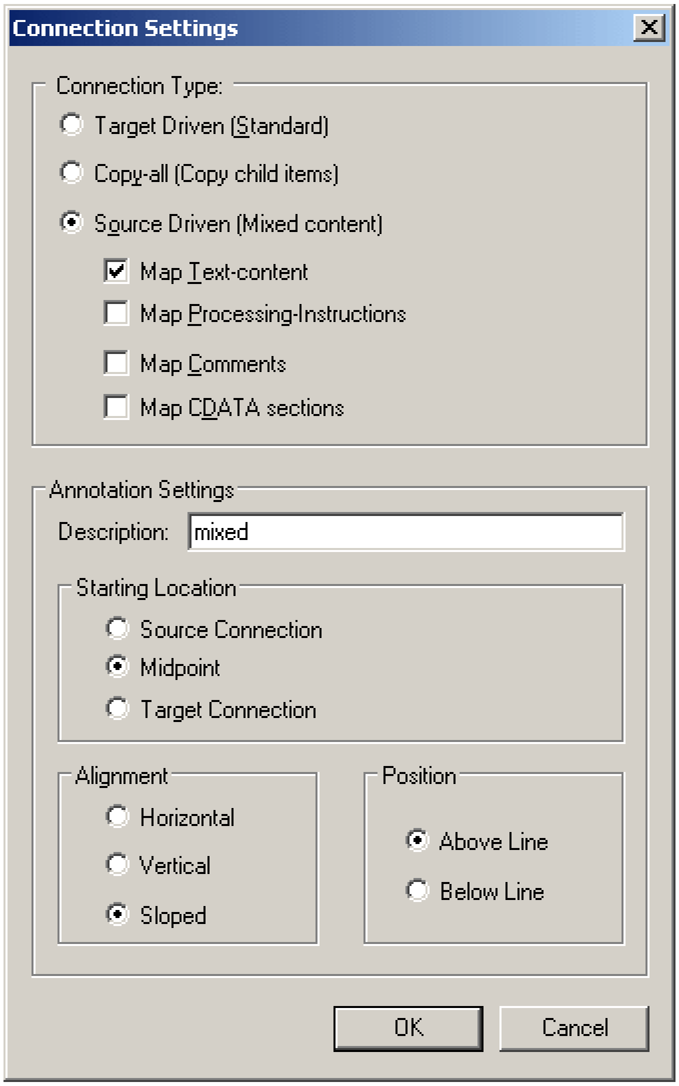
Source Driven (mixed content)

Changes the connector type to source driven / mixed content, and enables the selection of additional elements to be mapped. The additional elements have to be **child items** of the mapped item in the XML source file, to be able to be mapped. Please see Default settings: mapping mixed content for more information.

Connection settings:

Opens the Connection Settings dialog box in which you can define the specific (mixed content) settings of the current connector. Note that unavailable options are greyed out.

Please note that these settings also apply to **complexType** items which do not have any text nodes!



Connection Settings

Connection Type:

○ Target Driven (Standard)

○ Copy-all (Copy child items)

● Source Driven (Mixed content)

☑ Map Text-content

☐ Map Processing-Instructions

☐ Map Comments

☐ Map CDATA sections

Annotation Settings

Description: mixed

Starting Location

○ Source Connection

● Midpoint

○ Target Connection

Alignment

○ Horizontal

○ Vertical

● Sloped

Position

● Above Line

○ Below Line

Annotation settings:

Individual connectors can be labeled for clarity.

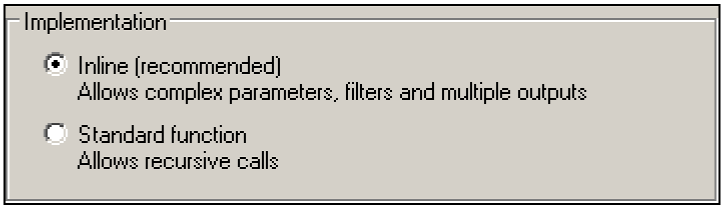
1. Double click a connector and enter the name of the connector in the Description field. This enables all the options in the Annotation Settings group.
2. Use the remaining groups to define the position and alignment of the label.

21.7 Function

Create user-defined function:
Creates a new user-defined function. Selecting this option creates and empty user-defined function, into which you insert the components you need. Please note that a single output function is automatically inserted when you define such a function, and that only one output function can be present in a user-defined function. Please see "Creating a user-defined function from scratch" for more information.

Create user-defined function based on selection:
Creates a new user-defined function based on the currently selected elements in the mapping window. Please note only one output component may exist amongst the selected components. Please see "Adding user-defined functions" for more information.

Function settings:
Opens the settings dialog box of the currently active user-defined function allowing you to change the current settings. Use this method to change the user-defined function type, i.e. double click the title bar of a user-defined function to see its contents, then select this menu option to change its type.



Implementation

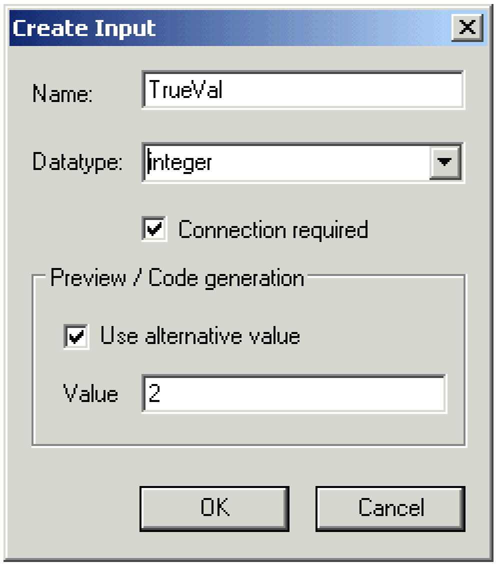
● Inline (recommended)
Allows complex parameters, filters and multiple outputs

○ Standard function
Allows recursive calls

Insert Input:
Inserts an "input" component into the mapping, or into a user-defined function.

If you are working in the main Mapping tab, the dialog box shown below is displayed. This type of input component allows you to:

- define an **override** value for the data that is being input by the current mapping input, and
- use this input component as a **parameter** in the command line execution of the compiled mapping.
  Please see "Input values, overrides and command line parameters" for more information.



Create Input

Name: TrueVal

Datatype: integer

☑ Connection required

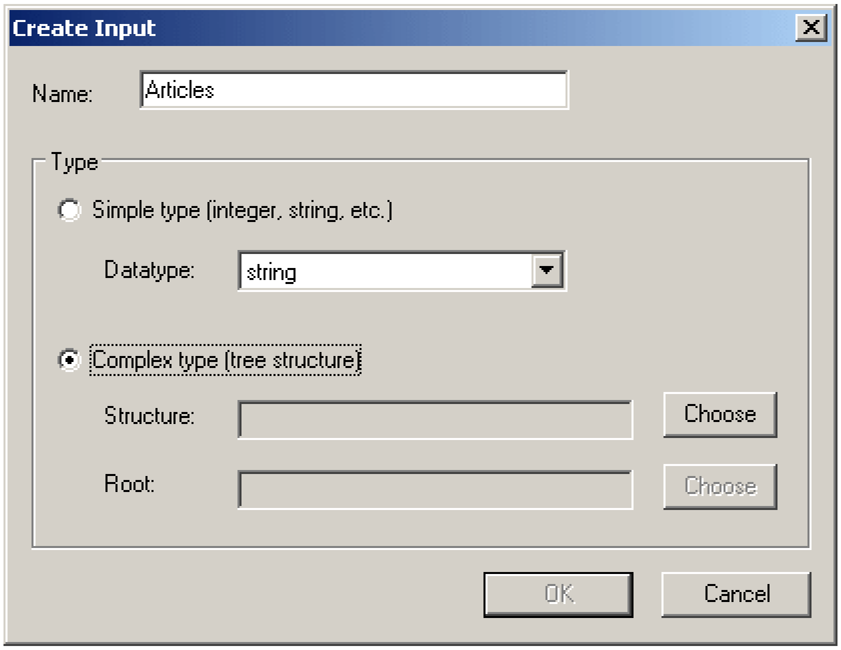
Preview / Code generation

☑ Use alternative value

Value 2

If you are working in a user-defined function tab, the dialog box shown below is displayed. This type of input component allows you to define:

- simple inputs, if this is a Standard user-defined function
- complex inputs, e.g. schema structures, if this is an Inline user-defined function.



Create Input

Name: Articles

Type

○ Simple type (integer, string, etc.)

Datatype: string

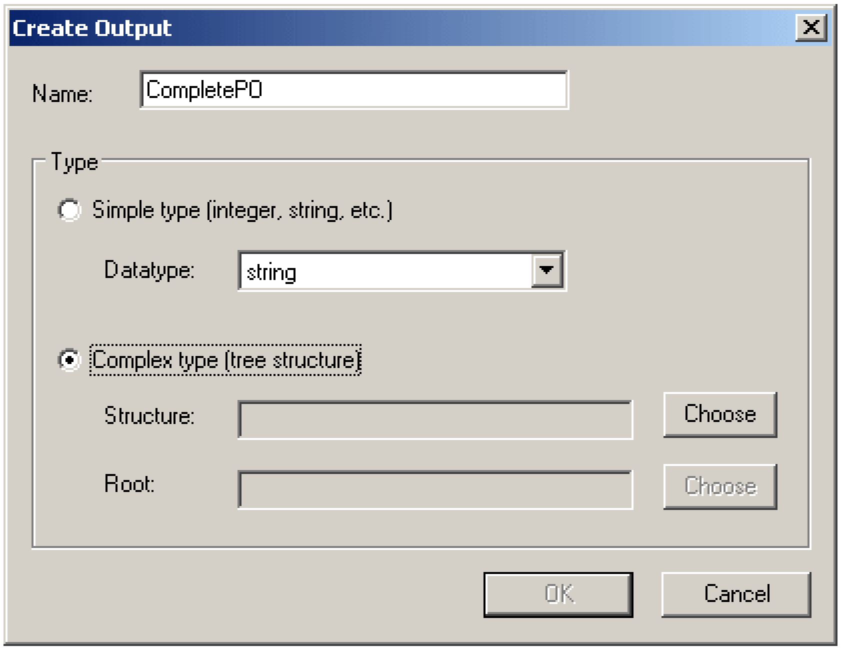
● Complex type (tree structure)

Structure:

Root:

Insert Output
Inserts an "Output" component into a user-defined function. In a user-defined function tab, the dialog box shown below is displayed. This type of input component allows you to define:

- simple outputs, if this is a Standard user-defined function
- complex outputs, e.g. schema structures, if this is an Inline user-defined function.



Name: CompletePO

Type

○ Simple type (integer, string, etc.)

Datatype: string

● Complex type (tree structure)

Structure:

Root:

21.8 Output

The **XSLT, XSLT2, XQuery, Java, C#, or C++**, options allow you to define the target language you want your code to be in. Note that the MapForce engine presents a preview of the mapping result, when you click the Output tab. Please see the MapForce engine section for more information.

Validate Output XML file
Validates the resultant XML file against the referenced schema.

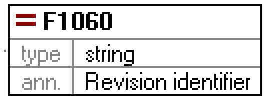
Save Output XML File
Saves the currently visible data in the Output tab.

Run SQL-script
Executes the pseudo-SQL select statements visible in the Output window, when the target component is a database.

21.9 View

Show Annotations

Displays XML schema annotations in the component window.
If the Show Types icon is also active, then both sets of info are show in grid form



F1060	
type	string
ann.	Revision identifier

Show Types

Displays the schema datatypes for each element or attribute.
If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in function header

Displays the library name in parenthesis in the function title.

Show Tips

Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

Show selected component connectors

Switches between showing:

- all mapping connectors, or
- those connectors relating to the currently selected components.

Show connectors from source to target

Switches between showing:

- connectors that are **directly** connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

Zoom

Opens the Zoom dialog box. You can enter the zoom factor numerically, or drag the slider to change the zoom factor interactively.

Status Bar

Switches the Status Bar, visible below the Messages window, on or off.

Library Window

Switches the Library window, containing all library functions, on or off.

Messages

Switches the Validation output window on, or off. When generating code the Messages output window is automatically activated to show the validation result.

Overview

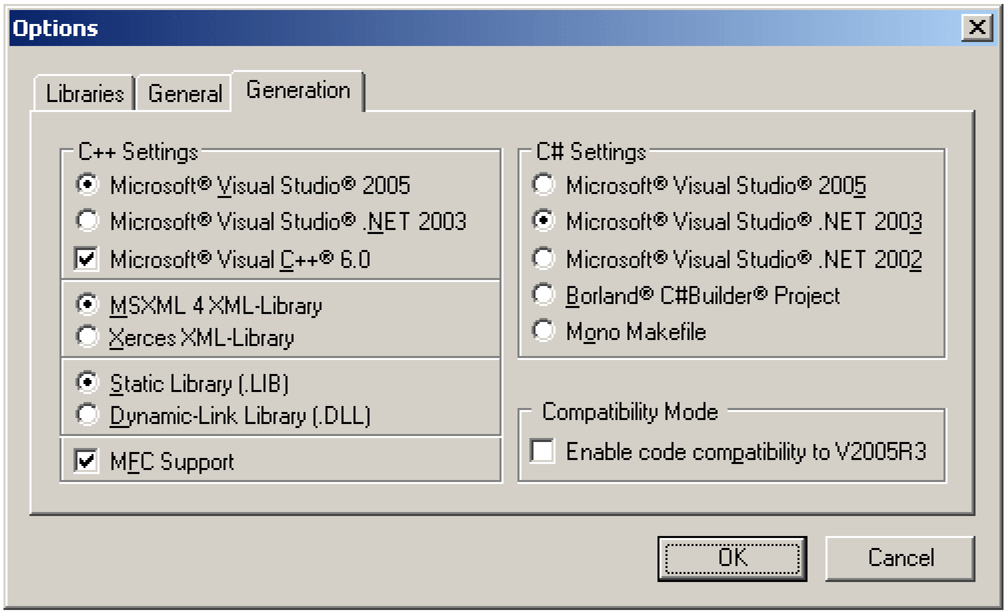
Switches the Overview window on, or off. Drag the rectangle to navigate your Mapping view.

21.10 Tools

Customize
The customize command lets you customize MapForce to suit your personal needs.

Options
Opens the Options dialog box through which you can:

- Add or delete user defined XSLT functions, or custom libraries.
- Define **general** settings, such as the default output encoding
- Define your specific compiler settings.



C++ Settings
● Microsoft® Visual Studio® 2005
○ Microsoft® Visual Studio® .NET 2003
☑ Microsoft® Visual C++® 6.0
● MSXML 4 XML-Library
○ Xerces XML-Library
● Static Library (.LIB)
○ Dynamic-Link Library (.DLL)
☑ MFC Support

C# Settings
○ Microsoft® Visual Studio® 2005
● Microsoft® Visual Studio® .NET 2003
○ Microsoft® Visual Studio® .NET 2002
○ Borland® C#Builder® Project
○ Mono Makefile

Compatibility Mode
□ Enable code compatibility to V2005R3

C++ Settings:
Defines the specific compiler settings for the C++ environment. Please note that the MFC Support check box must be activated for the source code to be compilable.

C# Settings:
Defines the specific compiler settings for the C# environment.

21.11 Help

Allows access to the Table of Contents and Index of the MapForce documentation, as well as Altova web site links. The Registration option opens the Altova Licensing Manager, which contains the licensing information for all of Altova products.

21.12 Oracle client installation

The instructions below describe the setting up of a new connection to an existing Oracle database somewhere on the local network. The Local net service name configuration wizard follows the same sequence when installing the Net Service during the initial installation of the Oracle client.

1. Select the menu option **Programs | Oracle - OraHome92 | Configuration and migration tools | Net Configuration Assistant**.
   This opens the Oracle Net Configuration Assistant.
2. Click the **Local Net Service Name configuration** radio button and click Next.
3. Click **Add** to add a new net service name and click Next.
4. Select the installed Oracle version, e.g. **Oracle 8i** or later... and click Next.
5. Enter the **Service Name** of the database you want to connect to e.g. TestDB and click Next. The database's service name is normally its global database name.
6. Select the **network protocol** used to access the database e.g. TCP, and click Next.
7. Enter the **Host name** of the computer on which the database is installed, and enter the port number if necessary. Click Next to continue.
8. Click the **Yes** radio button, to test the database connection, and click Next.
9. You can change the Login parameters if the test was not successful, by clicking the Change Login button, and trying again. Click Next to continue.
10. Enter the **Net Service Name** in the field of the same name, this can be any name you want. This is the name you will enter in the Database field, of the **Oracle login** dialog box in MapForce. Click Next to continue.
11. This completes the Net Service Name configuration. Click Next to close the dialog box.

Chapter 22

Code Generator

22 Code Generator

MapForce includes a built-in code generator which can automatically generate Java, C++ or C# class files from XML Schema definitions, text files, and databases. Mapping is not limited to simple one-to-one relationships; MapForce allows you to mix multiple sources and multiple targets, to map any combination of different data sources in a mixed environment.

The result of the code generation is fully-featured and complete software code which performs the mapping for you. You may insert the generated code into your own application, or extend it with your own functionality.

XML is not a full programming language in that it cannot be compiled or executed as a stand-alone binary executable file; rather XML documents must be bound to an external software application or runtime environment such as a business-to-business application or Web service.

Program coding is most commonly done through the use of high-level XML processing Application Program Interfaces (API) such as Microsoft MSXML and Apache Xerces Version 2.2.0 and higher, which are freely available for various programming languages.

22.1 Introduction to code generator

In the case of XML Schemas the MapForce code generator's default templates automatically generate class definitions corresponding to all declared elements or complex types which redefine any complex type in your XML Schema, preserving the class derivation as defined by extensions of complex types in your XML Schema, as well as all necessary classes which perform the mapping.

In the case of complex schemas which import schema components from multiple namespaces, MapForce preserves this information by generating the appropriate C#, or C++ namespaces or Java packages.

Additional code is implemented, such as functions which read XML files into a Document Object Model (DOM) in-memory representation, write XML files from a DOM representation back to a system file, as well as XML validation and transformation.

The output program code is expressed in C++, Java or C# programming languages.

C++
The C++ generated output uses either MSXML 4.0, or Apache Xerces 2.2 or later. Both MapForce and XMLSpy generate complete project and solution/workspace files for Visual C++ 6.0, Visual C++ 7.1 / Visual Studio .NET 2003 and 2005 directly.
.sln and .**vcproj** files are generated in addition to the **.dsw** /**.dsp** files for Visual Studio 6.0.

Please note:
When building C++ code for Visual Studio 2005 and using a Xerces library precompiled for Visual C++ 6.0, a compiler setting has to be changed in all projects of the solution:

1. Select all projects in the Solution Explorer.
2. [Project] | [Properties] | [Configuration Properties] | [C/C++] | [Language]
3. Select *All Configurations*
4. Change **Treat wchar_t** as **Built-in Type** to **No (/Zc:wchar_t-)**

C#
The generated C# code uses the .NET XML classes (System.Xml) and can be used from any .NET capable programming language, e.g. VB.NET, Managed C++, J# or any of the several languages that target the .NET platform. Project files can be generated for Visual Studio .NET, 2003, 2005 and Borland C#Builder.

Java
The generated Java output is written against the industry-standard Java API for XML Parsing (JAXP) and includes a JBuilder project file and an ANT build file.

Generated output in MapForce:

Generated output	**Location**	**MapForce**
Standard libraries	"Altova" folder	☑
Schema wrapper libraries	Schema name folder	☑
Database wrapper libraries	Database name folder	☑
		☑
Application		
Mapping application (complete app.)		☑
Compiling and executing, performs the defined mapping.		
		☑
Mapping application can nowextended by user, or be:		
	imported into own application	☑

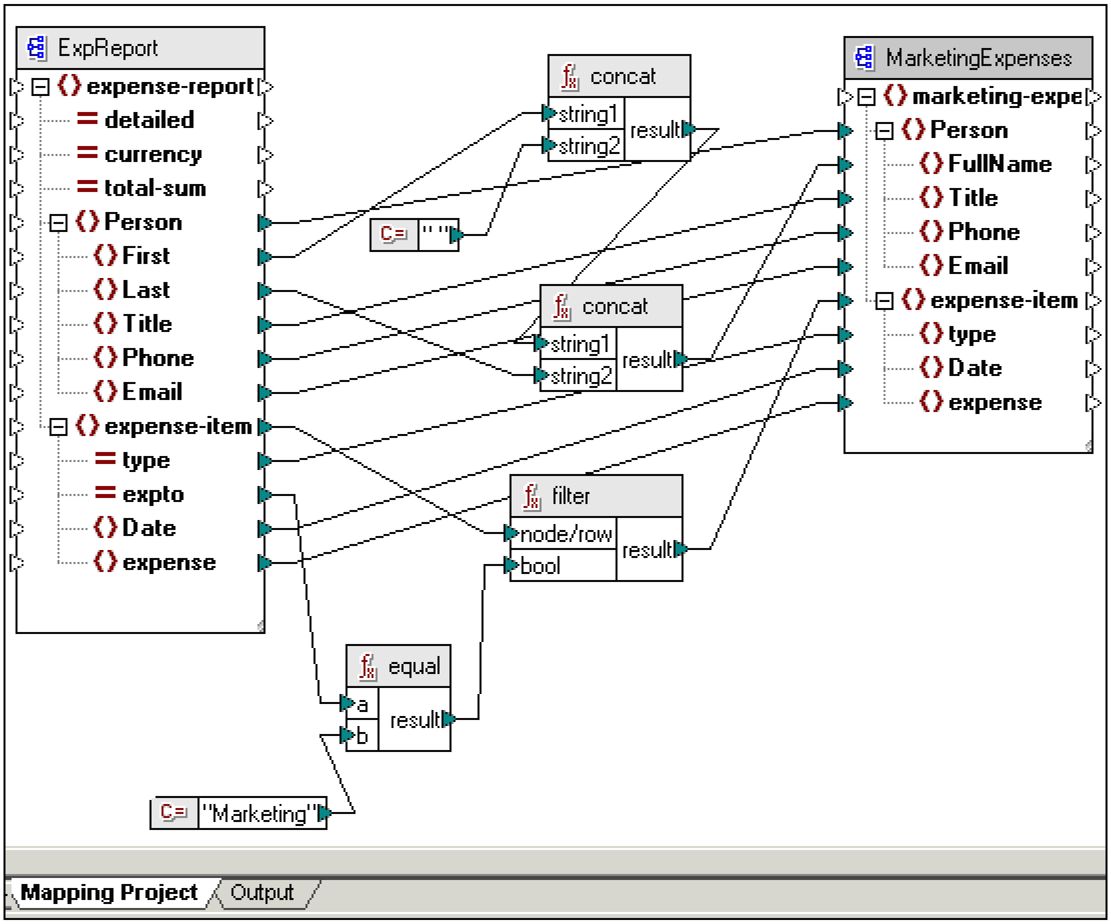
Code generator templates
Output code is completely customizable via a simple yet powerful template language which gives full control in mapping XML Schema built-in data-types to the primitive datatypes of a particular programming language.
It allows you to easily replace the underlying parsing and validating engine, customize code according to your company's writing conventions, or use different base libraries such as the Microsoft Foundation Classes (MFC) and the Standard Template Library (STL).

Built-in code generation frees software developers from the mundane task of writing low level infrastructure code, enabling them to focus on implementing critical business logic. By automatically generating a programming language binding, MapForce accelerates project development time from initial design to final implementation, resulting in substantial cost savings and time to market advantages.

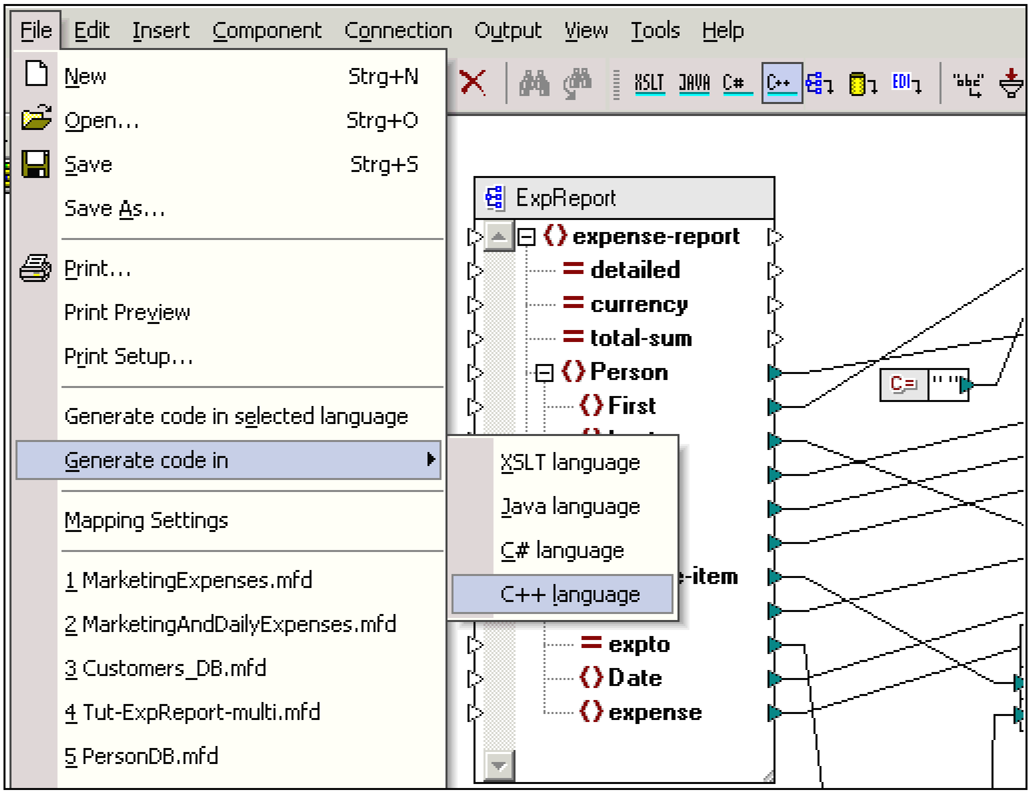
22.2 Generating program code

This example shows the general sequence that needs to be followed to generate program code from MapForce. The example uses the MarketingExpenses.mfd file available in the **..\MapforceExamples** folder. Please also see Integrating code in your application for more information.

1. Open the **MarketingExpenses.mfd** mapping file.



2. Select the menu option **File | Generate code in | C++**.



The **Browse for Folder** dialog box opens at this point.

3. Navigate to the folder that you want the code to be placed in, and click OK to confirm. A "C++ Code Generation completed" message appears.
4. Click OK to confirm the message.

The generated code is placed in subdirectories below the directory you specified, and contains all the necessary libraries etc. needed to compile and execute the mapping code.

The sequence shown here is repeated later in this document, with additional information on the build and compile process of each of the programming languages:

For more information please see the sections below:
Generating Java code
Generating C# code
Generating C++ code

22.2.1 Generating Java code

Prerequisites and default settings:

The generated mapforce application supports Java 5 code.

JDBC drivers have to be installed for you to compile Java code when mapping database data. Please see the section JDBC driver setup for more information.

The menu option **File | Mapping settings** defines the mapping project settings. The default settings are Application name=Mapping and Base Package Name=com.mapforce

JDBC specific settings can be viewed by clicking a database component, and selecting the menu option **Component | Component Settings**.

The table below shows the different applications that can be compiled in each of the environments:

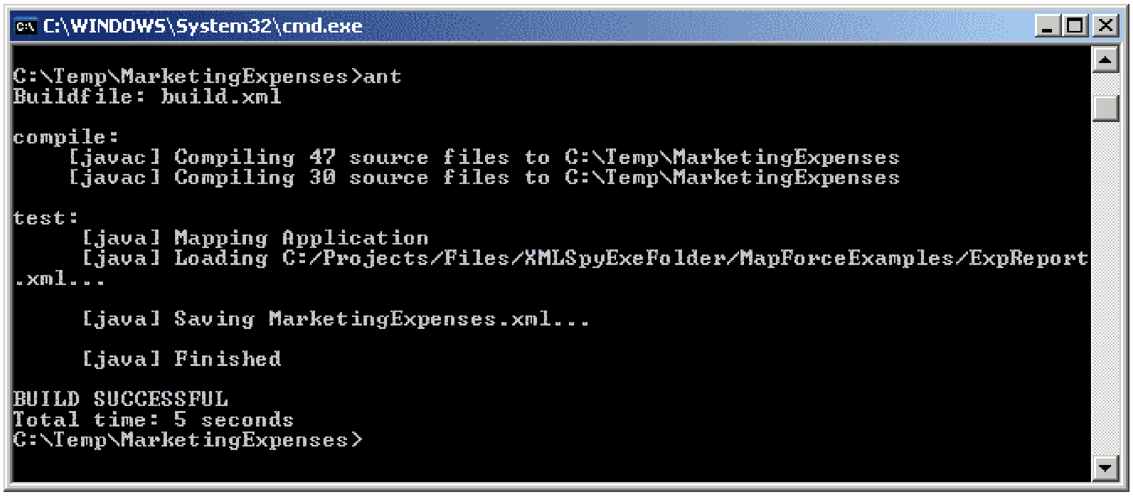
	Note	ANT	Sun ONE	JBuilder
MappingConsole.java	Console application	☑	☑	☑
MappingApplication.java	Dialog application	☐	☑	☑

To generate Java code in MapForce:

1. Select the menu option **File | Generate code in | Java**.
   You are then prompted for the target directory of the generated files.
2. Select the directory you want to place the files in, and click OK to confirm (eg. c:\codegen\Java).
   A "Java Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).

If you are using an **ANT** build script:

- Navigate to the Java subdirectory and execute "ant" (which automatically opens the **build.xml** file)
- This will **compile** and **execute** the Java code. The XML target instance file is automatically generated at the end of this sequence.



C:\WINDOWS\System32\cmd.exe

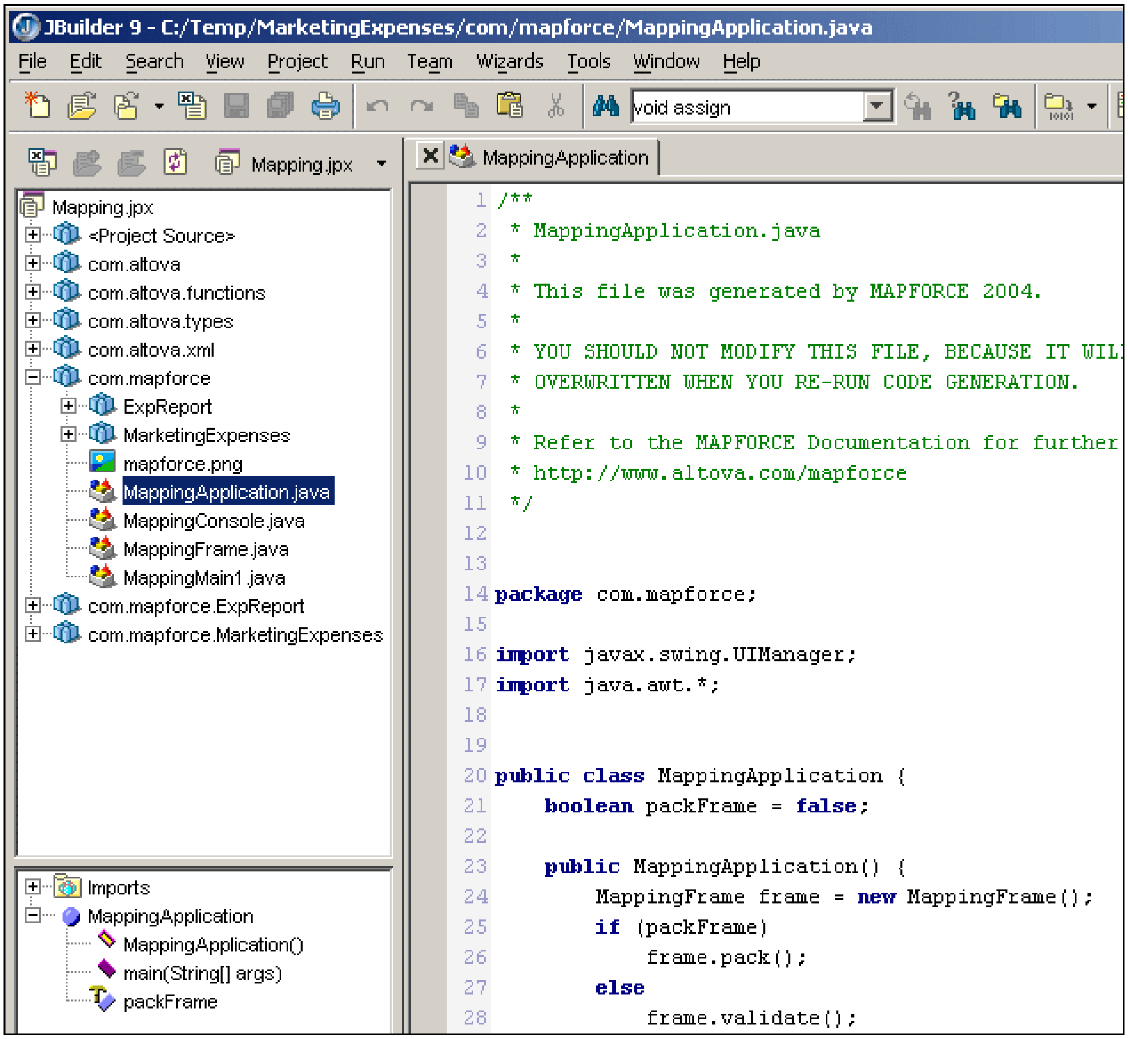
```
C:\Temp\MarketingExpenses>ant
Buildfile: build.xml

compile:
    [javac] Compiling 47 source files to C:\Temp\MarketingExpenses
    [javac] Compiling 30 source files to C:\Temp\MarketingExpenses

test:
     [java] Mapping Application
     [java] Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport
.xml...

     [java] Saving MarketingExpenses.xml...

     [java] Finished

BUILD SUCCESSFUL
Total time: 5 seconds
C:\Temp\MarketingExpenses>
```

For further information please see:
Generating Java code using JBuilder

Generating Java code using JBuilder

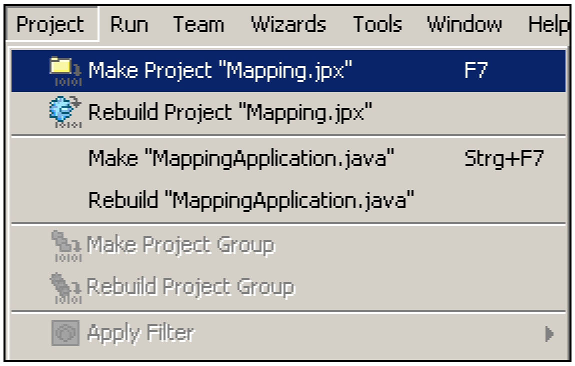
If you are using Borland **JBuilder**,

- Navigate to the Java subdirectory and open the **Mapping.jpx** file to compile the Java code,

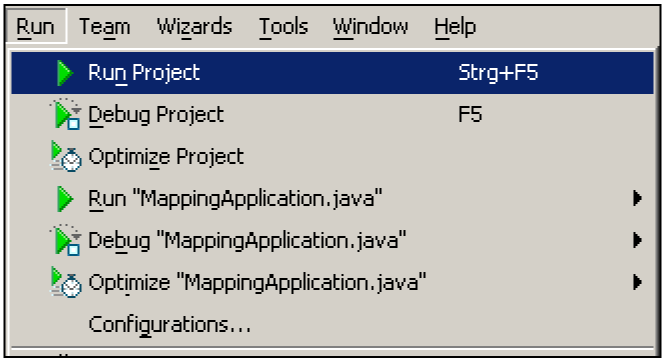


```
/**
 * MappingApplication.java
 *
 * This file was generated by MAPFORCE 2004.
 *
 * YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WIL
 * OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.
 *
 * Refer to the MAPFORCE Documentation for further
 * http://www.altova.com/mapforce
 */


package com.mapforce;

import javax.swing.UIManager;
import java.awt.*;


public class MappingApplication {
    boolean packFrame = false;

    public MappingApplication() {
        MappingFrame frame = new MappingFrame();
        if (packFrame)
            frame.pack();
        else
            frame.validate();
```

Select **Make Project "Mapping.jpx"** to compile the Java file.


Project Run Team Wizards Tools Window Help
Make Project "Mapping.jpx" F7
Rebuild Project "Mapping.jpx"
Make "MappingApplication.java" Strg+F7
Rebuild "MappingApplication.java"
Make Project Group
Rebuild Project Group
Apply Filter

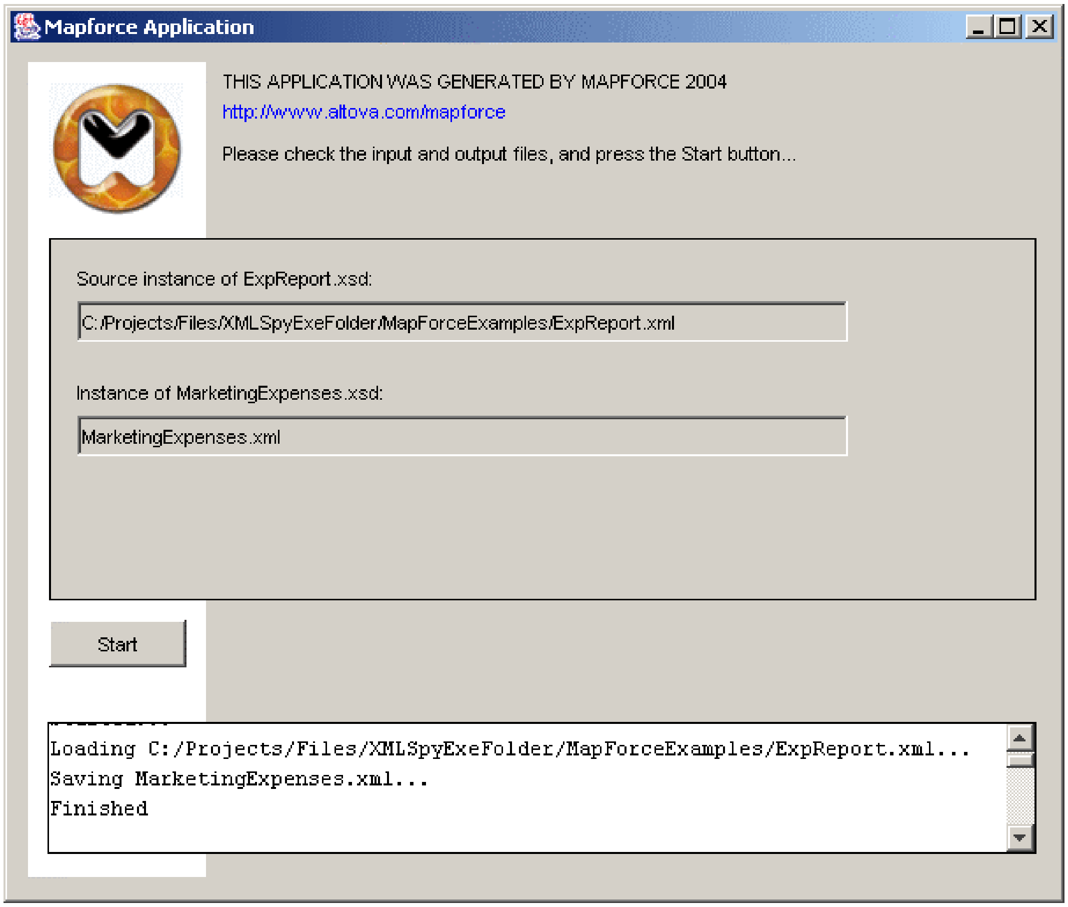
Select the menu option **Run | Run project.**



Execute either:

- MappingApplication, or
- MappingConsole
  In both cases the MarketingExpenses.xml target file is created.

MappingApplication



THIS APPLICATION WAS GENERATED BY MAPFORCE 2004

http://www.altova.com/mapforce

Please check the input and output files, and press the Start button...

Source instance of ExpReport.xsd:

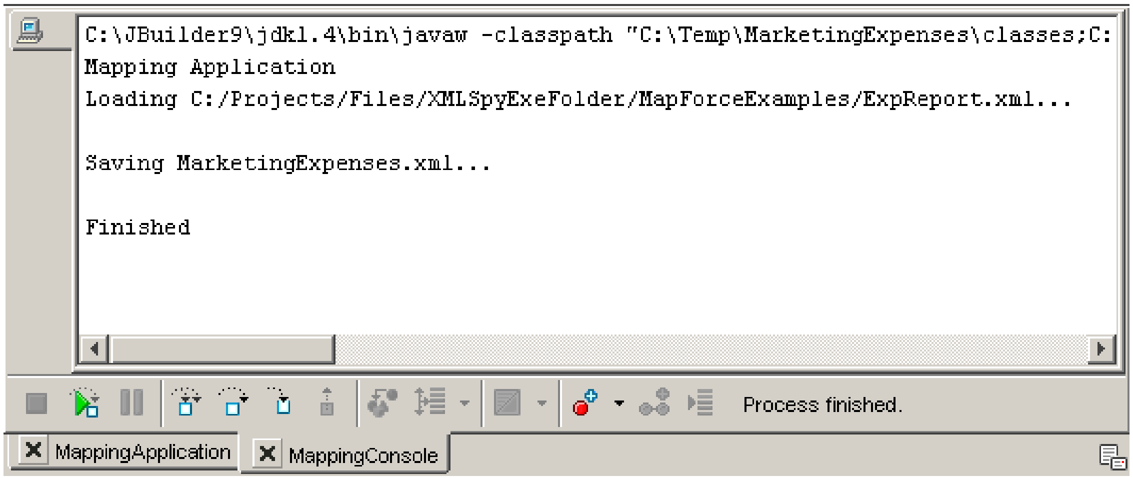
C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml

Instance of MarketingExpenses.xsd:

MarketingExpenses.xml

```
Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml...
Saving MarketingExpenses.xml...
Finished
```

MappingConsole
The screenshot below shows the mapping output in JBuilder.



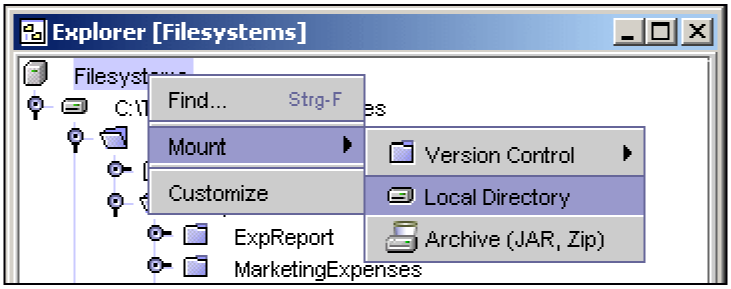
```
C:\JBuilder9\jdk1.4\bin\javaw -classpath "C:\Temp\MarketingExpenses\classes;C:
Mapping Application
Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml...

Saving MarketingExpenses.xml...

Finished
```

The **com** subdirectory contains the generated code in various subdirectories.

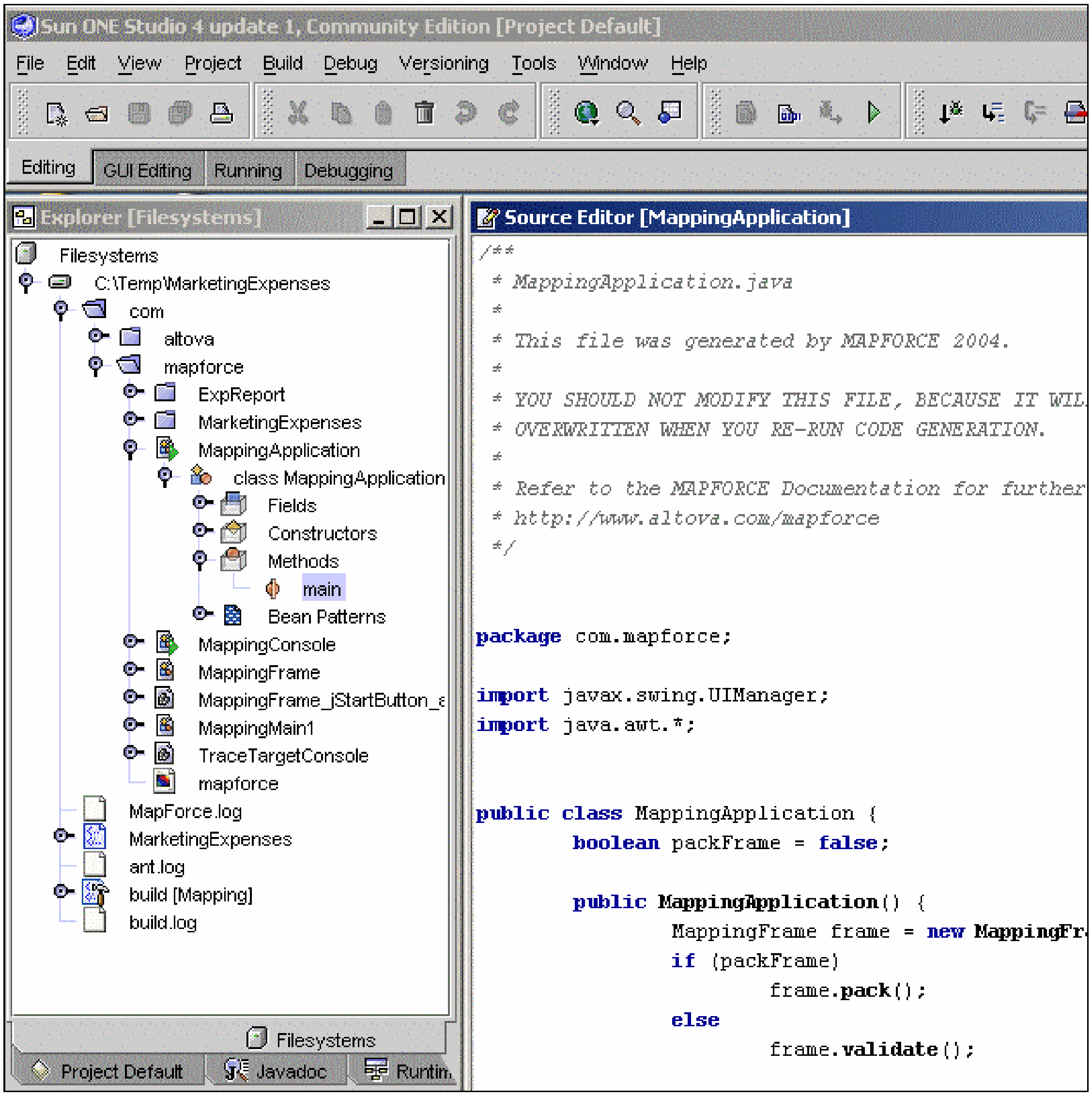
Generating Java code using Sun ONE Studio

If you are using **Sun ONE Studio:**

- Open Sun ONE studio.
- **Mount** the target directory you specified when you generated the Java code.

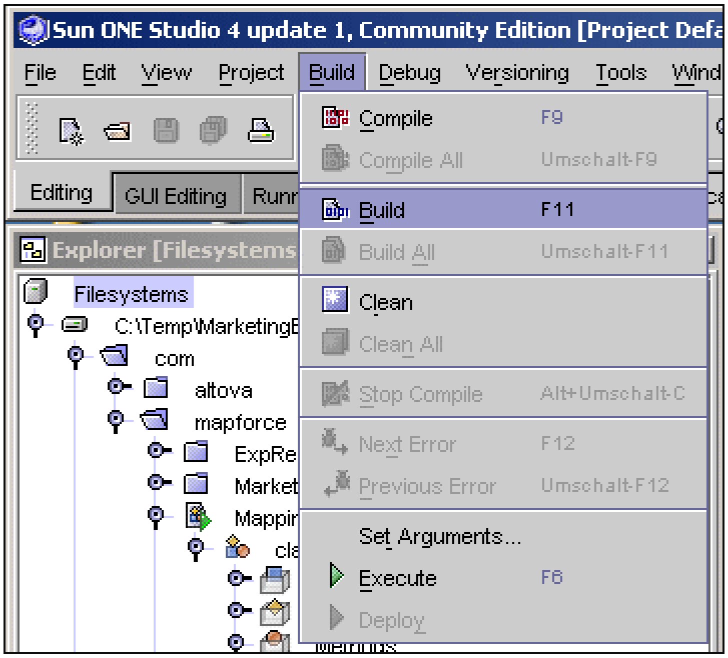


The Explorer window at left displays the source file structure.



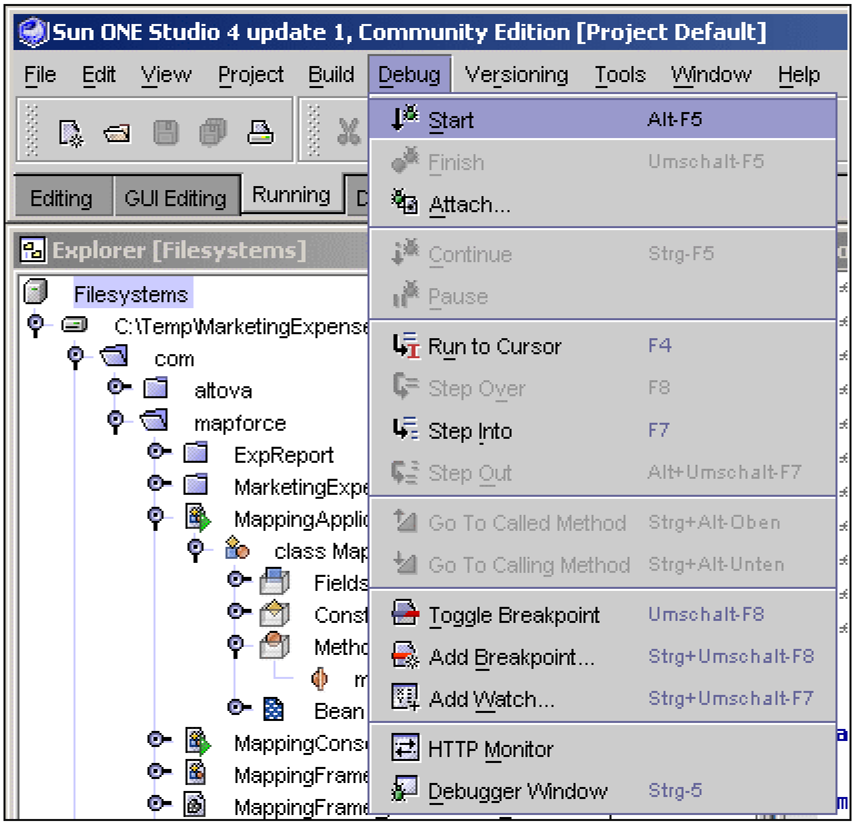
```
/**
 * MappingApplication.java
 *
 * This file was generated by MAPFORCE 2004.
 *
 * YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WIL
 * OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.
 *
 * Refer to the MAPFORCE Documentation for further
 * http://www.altova.com/mapforce
 */


package com.mapforce;

import javax.swing.UIManager;
import java.awt.*;


public class MappingApplication {
        boolean packFrame = false;

        public MappingApplication() {
                MappingFrame frame = new MappingFr
                if (packFrame)
                        frame.pack();
                else
                        frame.validate();
```

- Open either the **MappingApplication**, or **MappingConsole** folder, depending on which you want to edit and compile.
- Click the **main** method (in the Methods folder) to view the generated code.
- Select the menu option **Build | Build** to build the selected application, either:
    - MappingApplication, or
    - MappingConsole



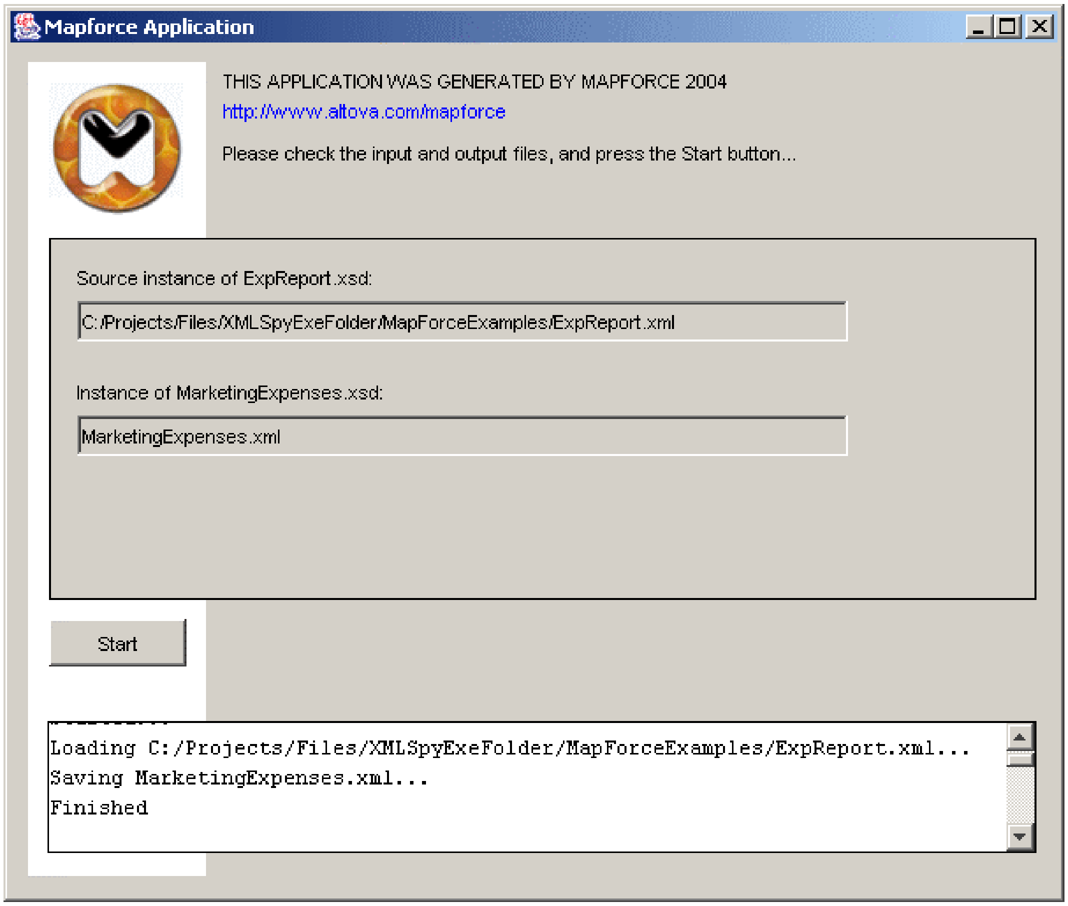
The build window displays a "successful" message when complete.

- Select the menu option **Debug | Start** to execute the selected application.



In both cases, MappingApplication or MappingConsole, the **MarketingExpenses.xml** target file is created.

MappingApplication



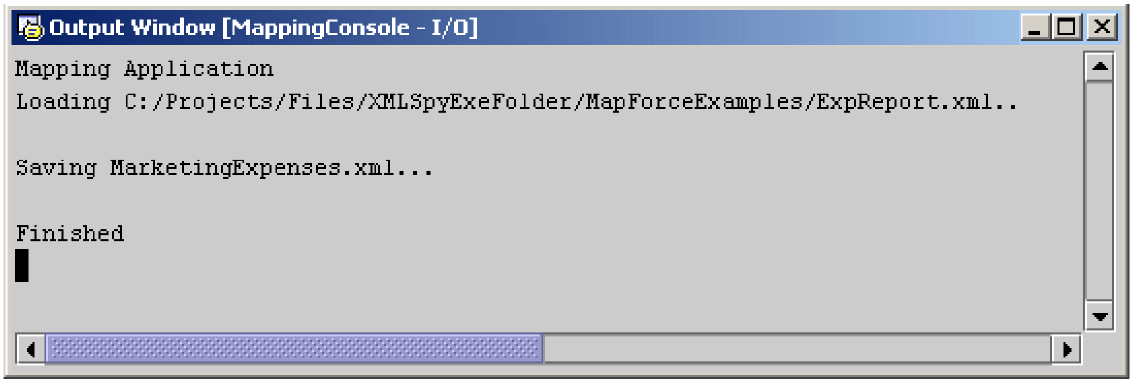
THIS APPLICATION WAS GENERATED BY MAPFORCE 2004
http://www.altova.com/mapforce
Please check the input and output files, and press the Start button...

Source instance of ExpReport.xsd:
C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml

Instance of MarketingExpenses.xsd:
MarketingExpenses.xml

```
Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml...
Saving MarketingExpenses.xml...
Finished
```

MappingConsole
The screenshot below shows the mapping console output in Sun ONE Studio.



```
Mapping Application
Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml..

Saving MarketingExpenses.xml...

Finished
```

22.2.2 Generating C# code

Prerequisites and default settings:

The menu option **File | Mapping settings** defines the mapping project settings. The default settings are Application name=Mapping.

Database specfic settings can be viewed by clicking a database component, and selecting the menu option **Component | Component Settings**.

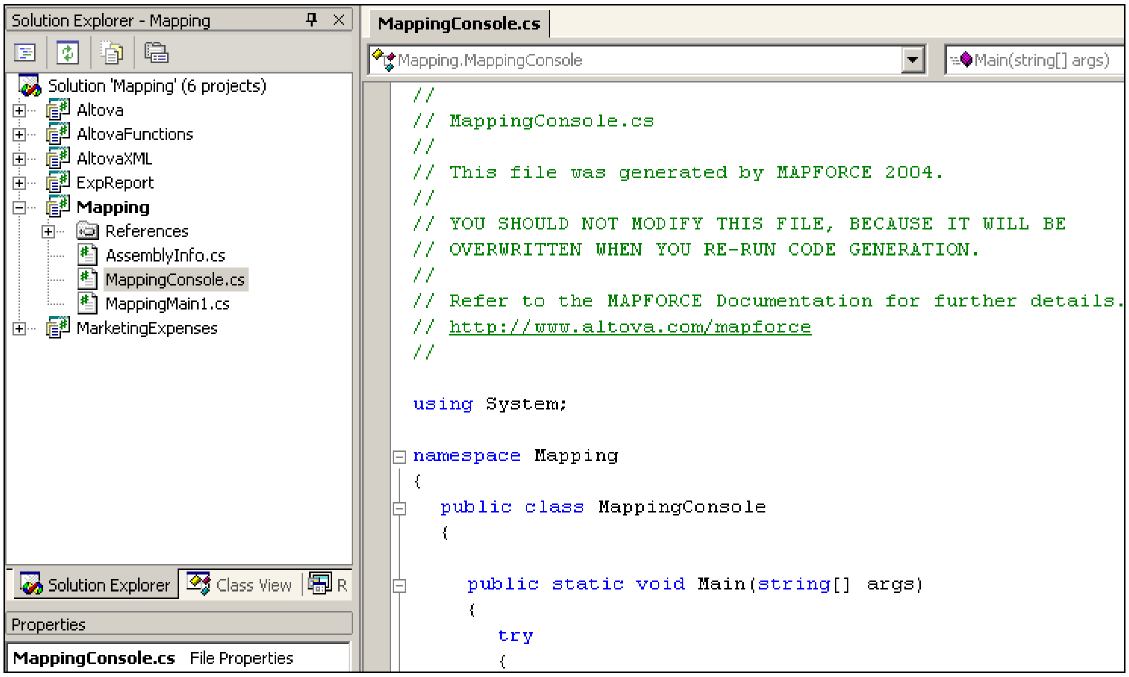
To generate C# code in MapForce:

1. Select the menu option **File | Generate code in | C# (sharp)**.
   You are then prompted for the target directory of the generated files.
2. Select the directory you want to place the files in, and click OK to confirm (eg. c:\codegen\C#).
   A "C# Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).

Folder **c:\codegen\C#\mapping**
You can compile the project in VS .NET 2002/2003/2005, or use the mono makefile.

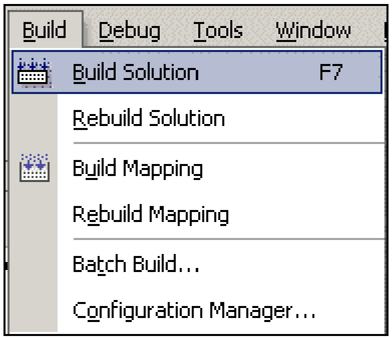
- Mapping.sln (for VS .NET 2002/2003/2005)
- The makefile is placed in the C# directory, if **Mono Makefile** was selected in the Generation tab of the (Tools) Options dialog box.

1. Navigate to the **Mapping** subdirectory, and open the Mapping solution file **Mapping.sln**

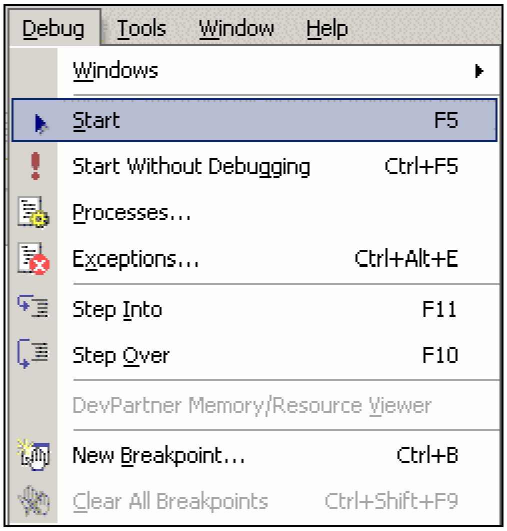


```
//
// MappingConsole.cs
//
// This file was generated by MAPFORCE 2004.
//
// YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WILL BE
// OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.
//
// Refer to the MAPFORCE Documentation for further details.
// http://www.altova.com/mapforce
//

using System;

namespace Mapping
{
  public class MappingConsole
  {

    public static void Main(string[] args)
    {
      try
      {
```

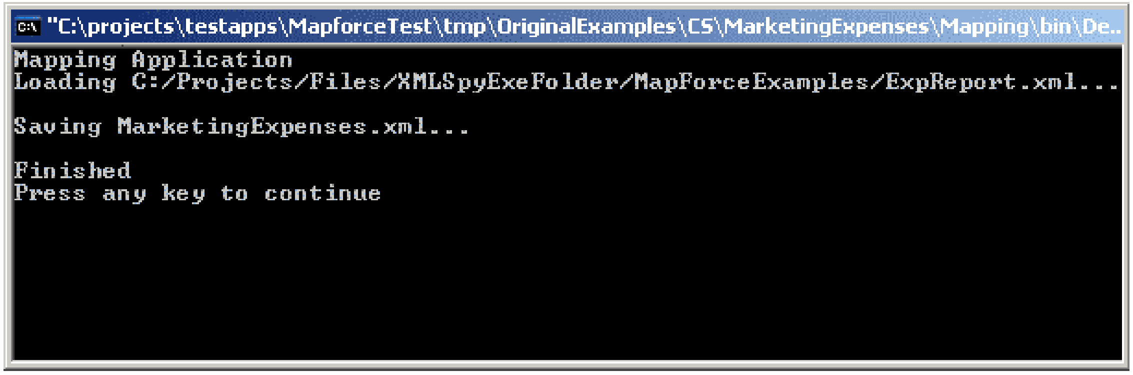
2. Select the menu option **Build | Build Solution** to compile the mapping project.



3. Select the menu option **Debug | Run** to start the application.



The mapping application is started and the target XML file is created.


"C:\projects\testapps\MapforceTest\tmp\OriginalExamples\CS\MarketingExpenses\Mapping\bin\De..

```
Mapping Application
Loading C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml...

Saving MarketingExpenses.xml...

Finished
Press any key to continue
```

22.2.3 Generating C++ code

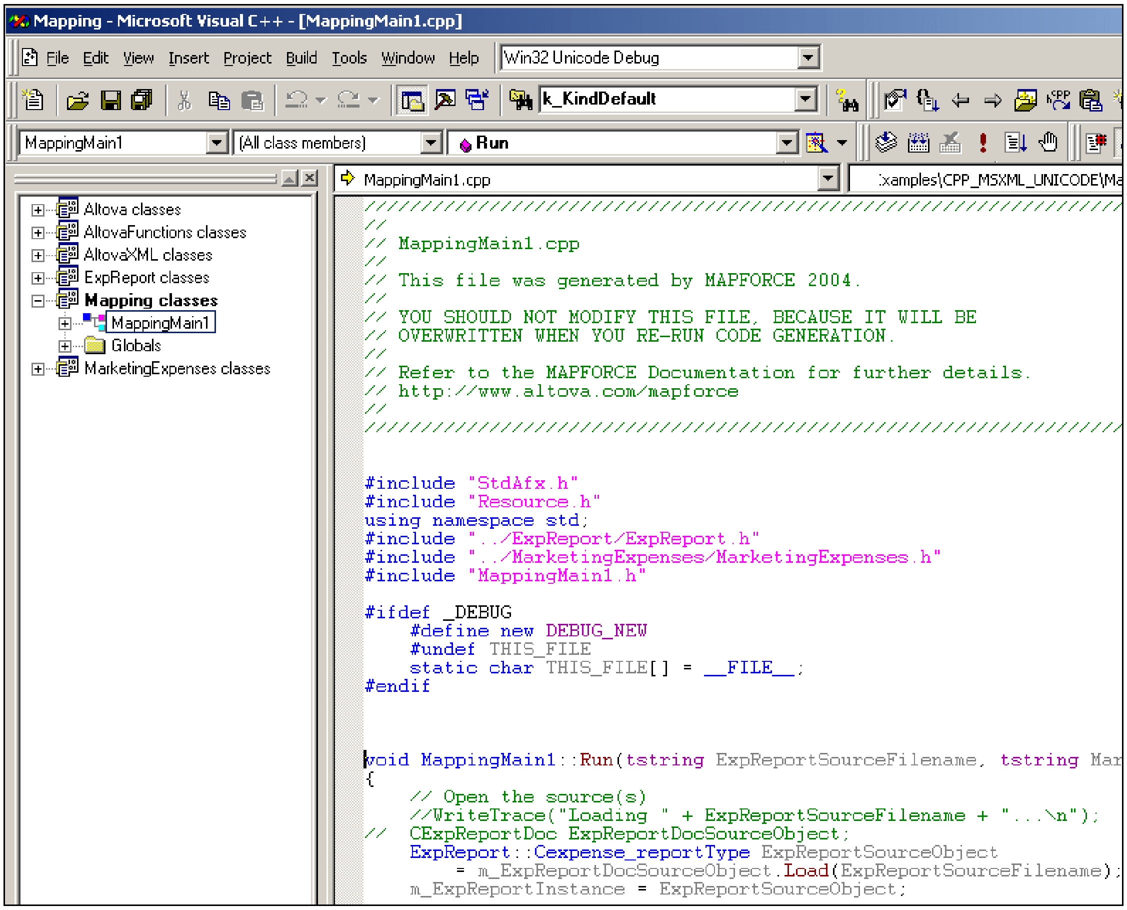
Prerequisites and default settings:

- C++ code generation for both MapForce and XMLSpy support Visual C++ 6.0, 7.1 / Visual Studio .NET 2003 and Visual Studio 2005 directly.
- **.sln** and **.vcproj** files are generated in addition to the **.dsw** /**.dsp** files for Visual Studio 6.0. The **Generation** tab under menu option **Tools | Options** allows you to choose the target IDE and code generation settings.
- The menu option **File | Mapping settings** defines the mapping project settings. The default settings are Application name=Mapping.

Database specfic settings can be viewed by clicking a database component, and selecting the menu option **Component | Component Settings**.

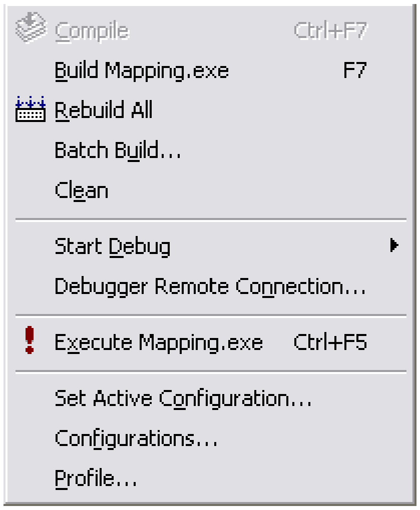
To generate C++ code in MapForce:

1. Select the menu option **File | Generate code in | C++**.
   You are then prompted for the target directory of the generated files.
2. Select the directory you want to place the files in, and click OK to confirm (eg. c:\codegen\C++).
   A "C++ Code generation completed" message appears when the process was successful. If not, an error message appears detailing the specific error(s).
3. Open the **mapping.dsw** or **mapping.sln** file in the **Mapping** subdirectory, i.e. c:\codegen\C++\mapping in Microsoft Visual C++.



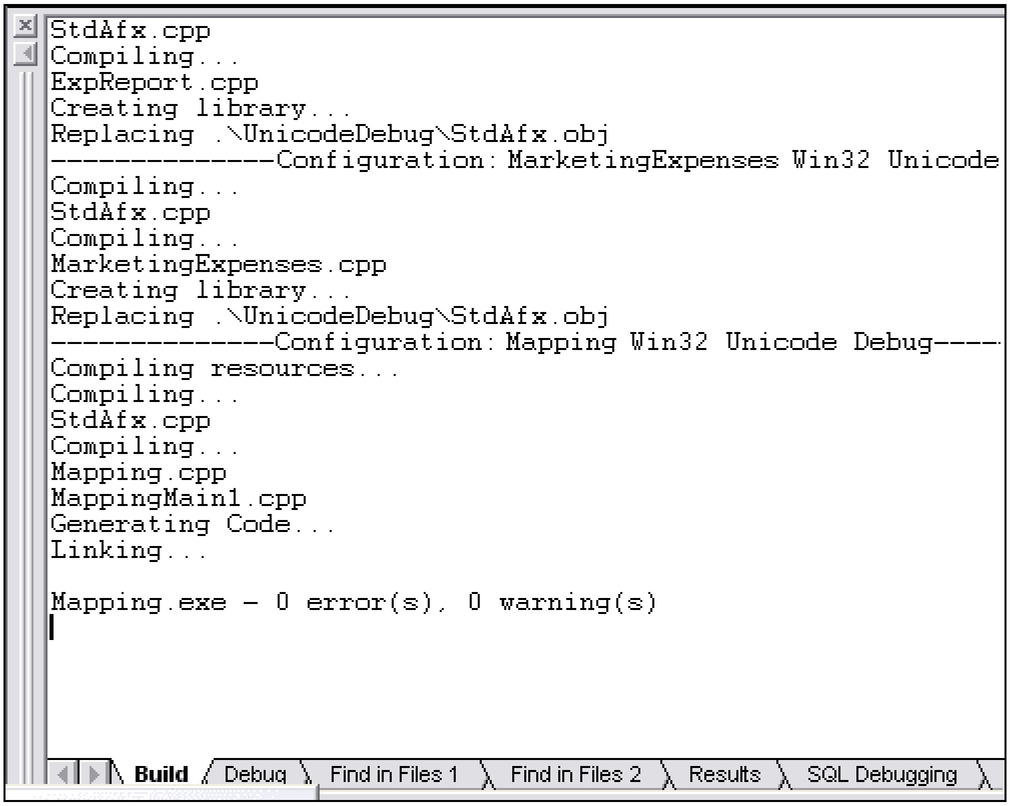
```
///////////////////////////////////////////////////////////////////
//
// MappingMain1.cpp
//
// This file was generated by MAPFORCE 2004.
//
// YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WILL BE
// OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.
//
// Refer to the MAPFORCE Documentation for further details.
// http://www.altova.com/mapforce
//
///////////////////////////////////////////////////////////////////


#include "StdAfx.h"
#include "Resource.h"
using namespace std;
#include "../ExpReport/ExpReport.h"
#include "../MarketingExpenses/MarketingExpenses.h"
#include "MappingMain1.h"

#ifdef _DEBUG
    #define new DEBUG_NEW
    #undef THIS_FILE
    static char THIS_FILE[] = __FILE__;
#endif


void MappingMain1::Run(tstring ExpReportSourceFilename, tstring Mar
{
    // Open the source(s)
    //WriteTrace("Loading " + ExpReportSourceFilename + "...\n");
//  CExpReportDoc ExpReportDocSourceObject;
    ExpReport::Cexpense_reportType ExpReportSourceObject
        = m_ExpReportDocSourceObject.Load(ExpReportSourceFilename);
    m_ExpReportInstance = ExpReportSourceObject;
```

4. Select the menu option **Build | Build Mapping.exe.**



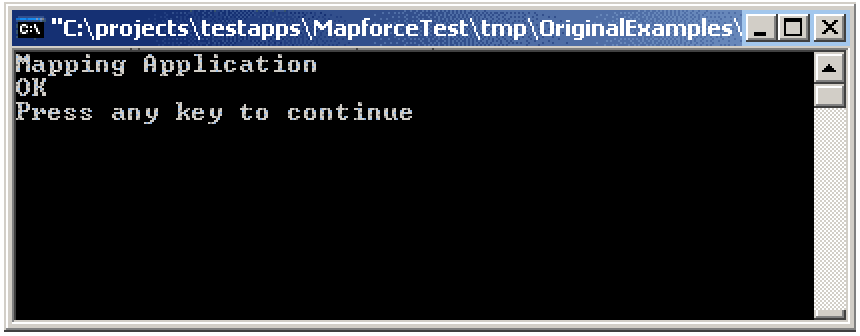
You can select from four different Build configurations:

Debug:	Debug Unicode Debug NonUnicode
Release:	Release Unicode Release Non Unicode



```
StdAfx.cpp
Compiling...
ExpReport.cpp
Creating library...
Replacing .\UnicodeDebug\StdAfx.obj
--------------Configuration: MarketingExpenses Win32 Unicode
Compiling...
StdAfx.cpp
Compiling...
MarketingExpenses.cpp
Creating library...
Replacing .\UnicodeDebug\StdAfx.obj
--------------Configuration: Mapping Win32 Unicode Debug----
Compiling resources...
Compiling...
StdAfx.cpp
Compiling...
Mapping.cpp
MappingMain1.cpp
Generating Code...
Linking...

Mapping.exe - 0 error(s), 0 warning(s)
```

5. Once the code has been built, execute the **Mapping.exe** program to map your data.



```
Mapping Application
OK
Press any key to continue
```

22.3 Code generation tips

Out-of-memory exceptions and how to resolve them:
It periodically occurs that mappings produce a very large amount of code, which might cause an out-of-memory exception during compilation. In Java this can be rectified by editing the **build.xml** file (used by ANT). The generated build.xml file is automatically created and placed in the folder you define when generating code.

Change line 6 from:

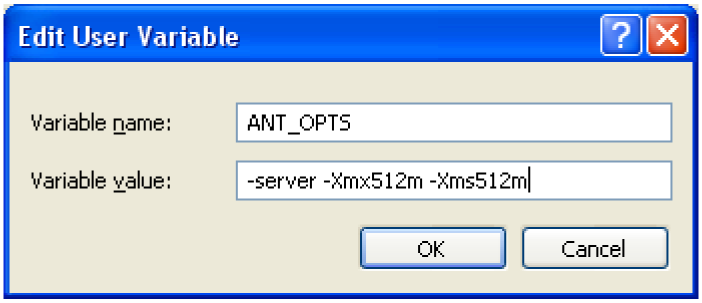
```
<javac srcdir="com/mapforce" destdir="."/>
```

to

<javac srcdir="com/mapforce" destdir="." **fork="true" memoryMaximumSize="80m"/**>

There is a related issue that might occur when compiling very large projects using ANT, which might also result in a java.lang.OutofMemory exception. To rectify this:

- Add the environmental variable **ANT_OPTS,** which sets specific ANT options such as the memory to be allocated to the compiler, and add values as shown below.



Variable name: ANT_OPTS
Variable value: -server -Xmx512m -Xms512m

Reserving method names:
When customizing code generation using the supplied spl files, it might be necessary to reserve method names. To do this:

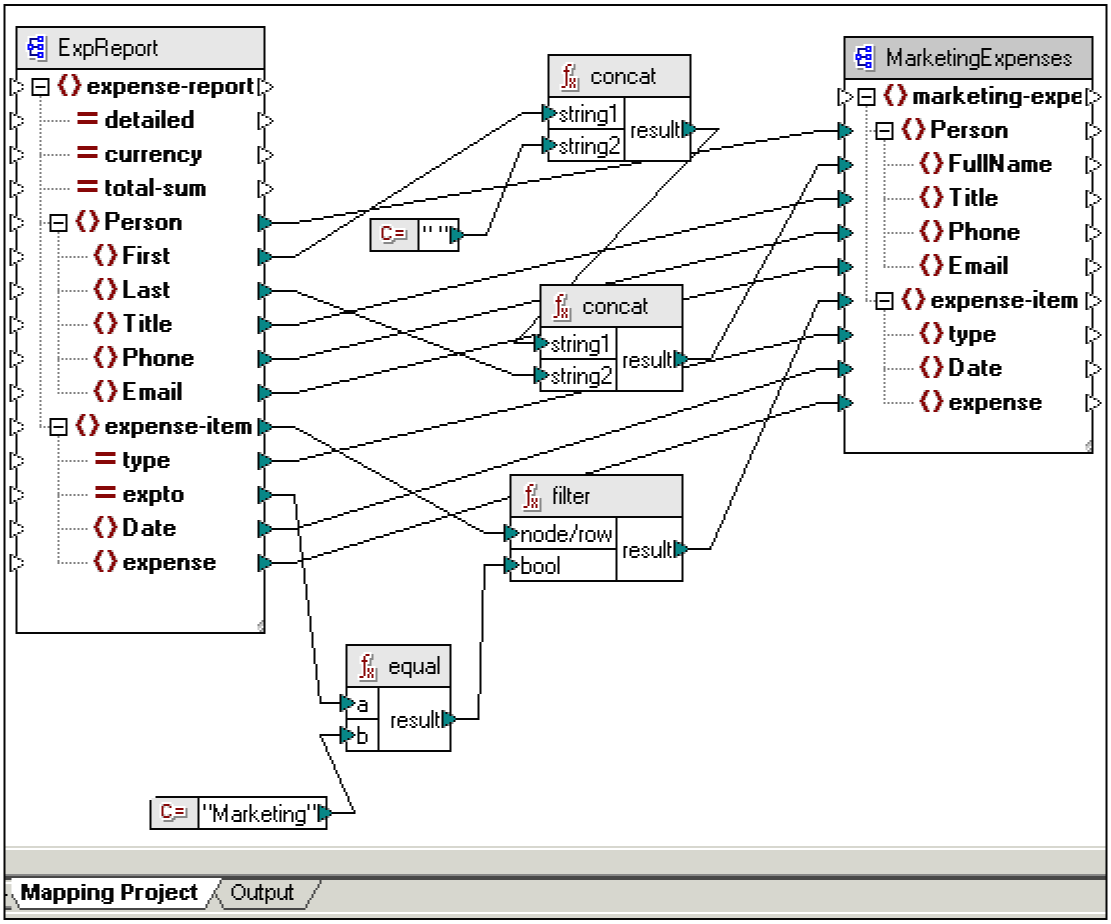
C# and **Java**:
1. Navigate to the program installation directory e.g. c:\Program Files\Altova\MapForce2006\spl\.
2. Open either the **cs.spl**, or **java.spl** file, and insert a new line into the reserve section e.g. reserve "myReservedWord"
3. Regenerate the program code.

C++
1. Navigate to the **cpp** subdirectory e.g. c:\Program Files\Altova\MapForce2006\spl\cpp\.
2. Open the **settings.spl** file and insert a new line into the reserve section e.g. reserve "myReservedWord".
3. Regenerate the program code.

XML Schema support:
Please note that the attribute nillable="true" is not supported during the code generation process.

22.4 Code generation mapping example

The mapping example shown below, uses the Marketing Expenses mapping project (MarketingExpenses.mfd) available in the **..\MapforceExamples** folder. Please also see Integrating code in your application for more information.



- The code below shows the **source** file path, as well as the **target** XML file produced by the mapping. Multiple source files can appear, however, only one target file may be associated.

```
"C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
                           "MarketingExpenses.xml" );
```

- If **multiple targets** exist, then the MappingMain section, shown below, is **repeated** for each target.

```
MappingMain1 MappingMain1Object = New MappingMain1();
MappingMain1Object.registerTraceTarget(ttc);
MappingMain1Object.run(
  "C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
  "MarketingExpenses.xml" );
```

- Extra error handling code can be inserted under the Exception section:

```
catch (Exception e), or
catch (CAltovaException& e)
```

The code snippets below, are the mapping code generated for each specific programming language.

Java (MappingConsole.java)

```
public static void main(String[] args) {
  try {
   System.out.println("Mapping Application");
   TraceTargetConsole ttc = New TraceTargetConsole();

   MappingMain1 MappingMain1Object = New MappingMain1();
   MappingMain1Object.registerTraceTarget(ttc);
   MappingMain1Object.run(
     "C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
     "MarketingExpenses.xml" );


   System.out.println("Finished");
  } catch (Exception e) {
   System.out.print("ERROR:");
   System.out.println( e.getMessage() );
   e.printStackTrace();
   System.exit(1);
  }
 }
```

C# (MappingConsole.cs)

```
public static void Main(string[] args)
  {
   try
   {
    Console.Out.WriteLine("Mapping Application");
    TraceTargetConsole ttc = new TraceTargetConsole();

    MappingMain1 MappingMain1Object = new MappingMain1();
    MappingMain1Object.RegisterTraceTarget(ttc);
    MappingMain1Object.Run(
     "C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml",
     "MarketingExpenses.xml");


    Console.Out.WriteLine("Finished");
   }
   catch (Exception e)
   {
    Console.Out.Write("ERROR: ");
    Console.Out.WriteLine( e.Message );
    Console.Out.WriteLine( e.StackTrace );
    System.Environment.Exit(1);
   }
  }
```

C++ (Mapping.cpp)

```
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
 tcout << _T("Mapping Application") << endl;

 try
 {
  CoInitialize(NULL);
  {
   MappingMain1 MappingMain1Object;
   MappingMain1Object.Run(
    _T("C:/Projects/Files/XMLSpyExeFolder/MapForceExamples/ExpReport.xml"),
    _T("MarketingExpenses.xml"));
  }
  CoUninitialize();

  tcout << _T("OK") << endl;
  return 0;
 }
 catch (CAltovaException& e)
 {
  tcerr << _T("Error: ") << e.GetInfo().c_str() << endl;
  return 1;
 }
 catch (_com_error& e)
 {
  tcerr << _T("COM-Error from ") << (TCHAR*)e.Source() << _T(":") << endl;
  tcerr << (TCHAR*)e.Description() << endl;
  return 1;
 }
 catch (std::exception& e)
 {
  cerr << "Exception: " << e.what() << endl;
  return 1;
 }
 catch (...)
 {
  tcerr << _T("Unknown error") << endl;
  return 1;
 }
}
```

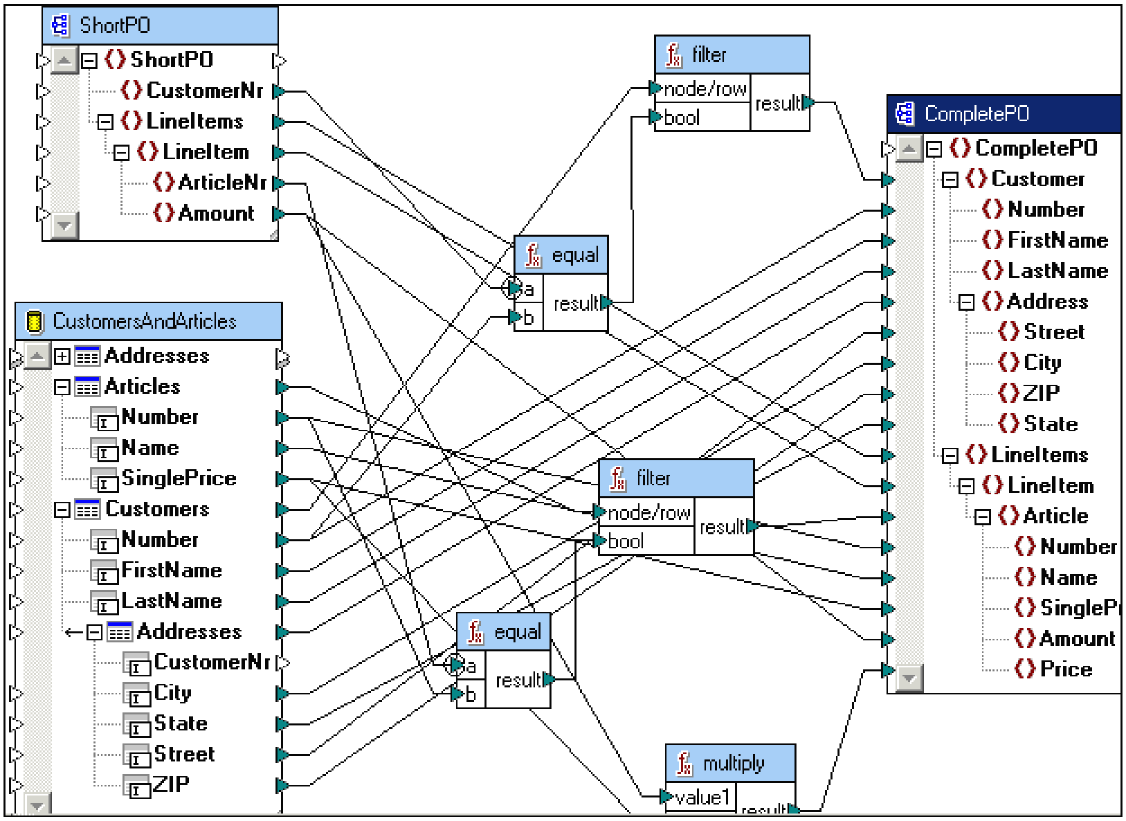
22.5 Integrating MapForce code in your application

MapForce generated code can be integrated, or adapted to your specific application, even though the result of code generation is a complete and fully-functioning application. The mapping project visible below, **DB_CompletePO.mfd**, is available in the ..**MapforceExamples** folder.

Please also see the section Generating program code for information on how the generated code is produced.

This section describes how to:

- **Modify** the source and target files of a mapping project
- Use an **XML input stream** as a data source, and
- Where to add your own **error handling** code



This example consists of two source and one target files:

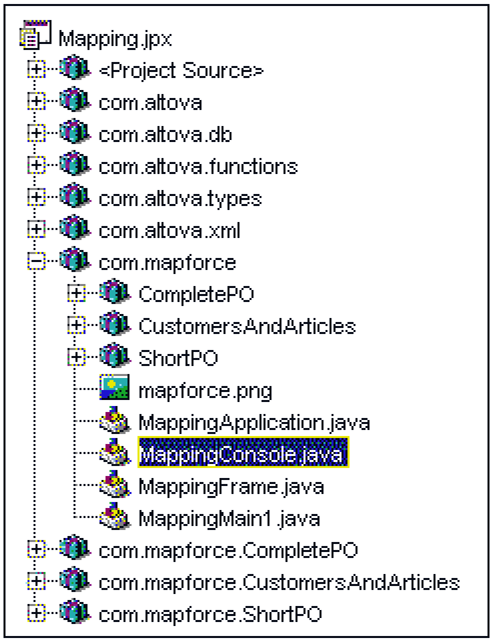
- ShortPO.xml as a **source XML** file
- CustomersAndArticles.mdb as a **source database**, and
- CompletePO.xml as the **target XML** file.

22.5.1 MapForce code in Java applications

This example assumes that you are using **Borland JBuilder** as your Java environment.

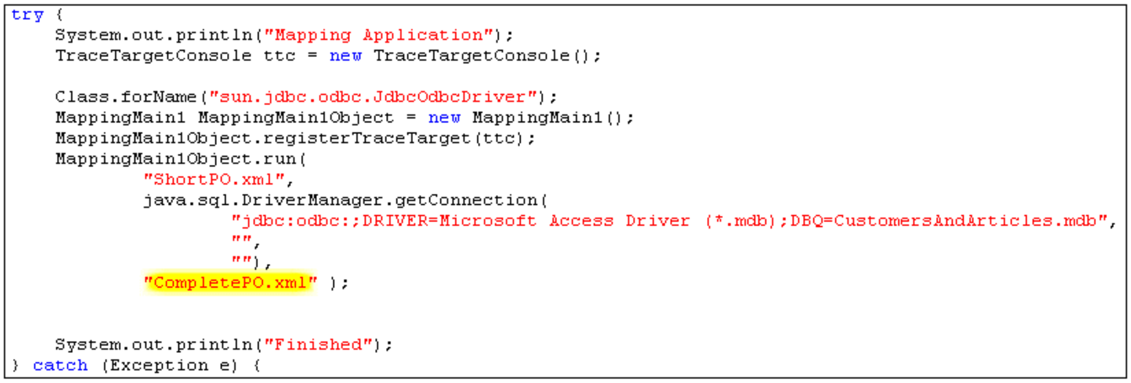
Having generated the Java code in MapForce, and defined a folder named "output" as the output folder,

1. Navigate to the output**Mapping** folder, and open the **Mapping.jpx** project file.
2. Double click the **MappingConsole.java** file.



```
Mapping.jpx
├─ <Project Source>
├─ com.altova
├─ com.altova.db
├─ com.altova.functions
├─ com.altova.types
├─ com.altova.xml
├─ com.mapforce
│   ├─ CompletePO
│   ├─ CustomersAndArticles
│   ├─ ShortPO
│   ├─ mapforce.png
│   ├─ MappingApplication.java
│   ├─ MappingConsole.java
│   ├─ MappingFrame.java
│   └─ MappingMain1.java
├─ com.mapforce.CompletePO
├─ com.mapforce.CustomersAndArticles
└─ com.mapforce.ShortPO
```

A snippet of the code is shown below.



```
try {
    System.out.println("Mapping Application");
    TraceTargetConsole ttc = new TraceTargetConsole();

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    MappingMain1 MappingMain1Object = new MappingMain1();
    MappingMain1Object.registerTraceTarget(ttc);
    MappingMain1Object.run(
            "ShortPO.xml",
            java.sql.DriverManager.getConnection(
                "jdbc:odbc:;DRIVER=Microsoft Access Driver (*.mdb);DBQ=CustomersAndArticles.mdb",
                "",
                ""),
            "CompletePO.xml" );


    System.out.println("Finished");
} catch (Exception e) {
```

Please note that the path names in the generated source code have been deleted for the sake of clarity.

Looking at **MappingMain1Object.run**:
All parameters passed to the **run** method, **except for** the last one (CompletePO.xml), are source files. Please ensure that the last parameter is the target file.

In this example:

the **source** files are:

- XML file: Short.PO.xml

- Database file: CustomerAndArticles.mdb including the connection string.
  The **two empty parameters** **" "** following the initial database parameter, are intended for the **Username** and **Password** (in clear text) for those databases where this data is necessary.

the **target** file is:

- CompletePO.xml

To define your own source or target files:

- Directly edit the parameters passed to the run method of MappingMain1Object.

To use an XML input stream as the XML data source:

- Navigate to the run method declaration in the code, and configure the specific parameters there.

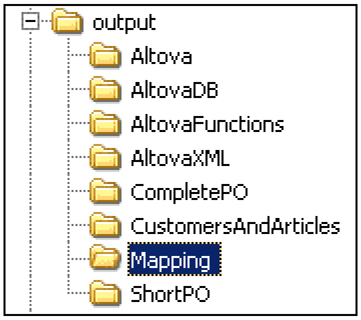
To add extra error handling code:

- Edit the code below the **`catch`** `(Exception e)` code.

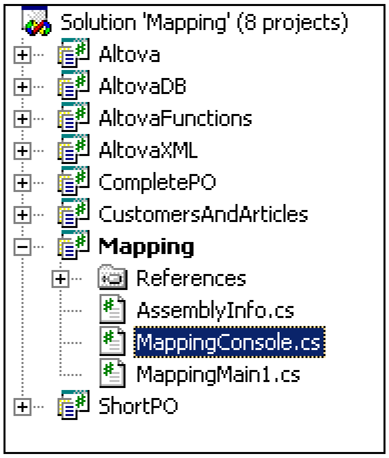
22.5.2 MapForce code in C# applications

Having generated the C# code in MapForce, and defined a folder named "output" as the output folder,

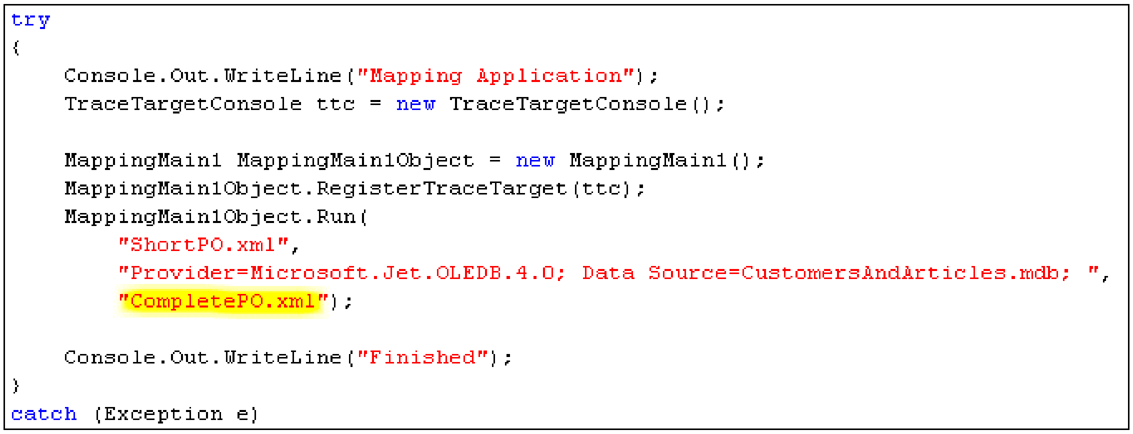
1. Navigate to the output**Mapping** folder, and open the **Mapping.sln** file.



2. Double click the **MappingConsole.cs** file.



A snippet of the code is shown below.



```
try
{
    Console.Out.WriteLine("Mapping Application");
    TraceTargetConsole ttc = new TraceTargetConsole();

    MappingMain1 MappingMain1Object = new MappingMain1();
    MappingMain1Object.RegisterTraceTarget(ttc);
    MappingMain1Object.Run(
        "ShortPO.xml",
        "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=CustomersAndArticles.mdb; ",
        "CompletePO.xml");

    Console.Out.WriteLine("Finished");
}
catch (Exception e)
```

Please note that the path names in the generated source code have been deleted for the sake of clarity.

Looking at **MappingMain1Object.Run**:
All parameters passed to the **run** method, **except for** the last one (CompletePO.xml), are source files. Please ensure that the last parameter is the target file.

In this example:

the **source** files are:

- XML file: Short.PO.xml
- Database file: CustomerAndArticles.mdb including the connection string

the **target** file is:

- CompletePO.xml

To define your own source or target files:

- Directly edit the parameters passed to the run method of MappingMain1Object.

To use an XML input stream as the XML data source:

- Navigate to the run method declaration in the code, and configure the specific parameters there.

To add extra error handling code:

Edit the code below the `catch (Exception e)` code.

Mono makefile

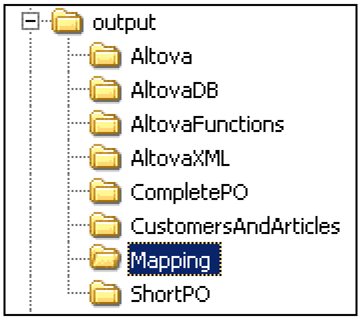
If you generated C# code and specified a **mono makefile** in the Generation tab of the **Tools | Options** dialog box:

- The makefile is placed in the ..\output folder
- Edit the **MappingConsole.cs** file in the output**Mapping** folder as mentioned above.

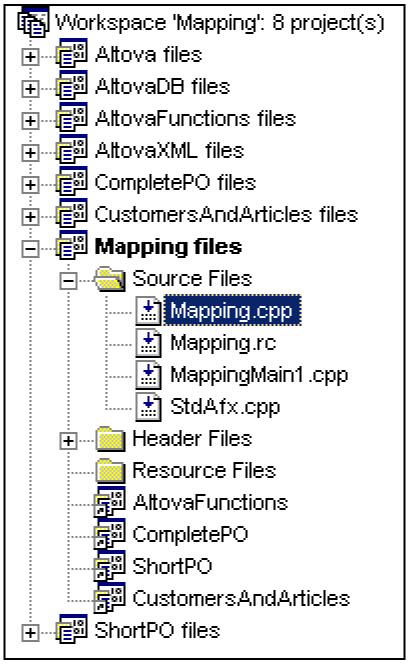
22.5.3 MapForce code in C++ applications

Having generated the C++ code in MapForce, and defined a folder named "output" as the output folder,

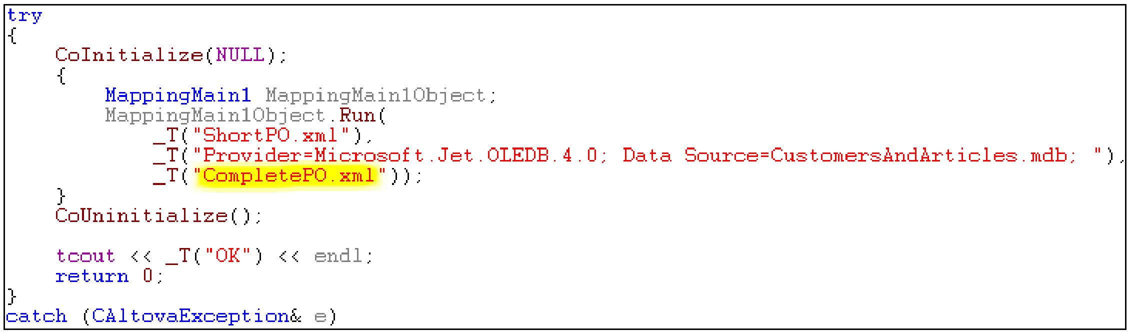
1. Navigate to the output**Mapping** folder, and open the workspace file **mapping.dsw** in MS Visual Studio 6.0



2. Double click the **Mapping.cpp** file to open the mapping project file.



A snippet of the code is shown below.



```
try
{
    CoInitialize(NULL);
    {
        MappingMain1 MappingMain1Object;
        MappingMain1Object.Run(
            _T("ShortPO.xml"),
            _T("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=CustomersAndArticles.mdb; "),
            _T("CompletePO.xml"));
    }
    CoUninitialize();

    tcout << _T("OK") << endl;
    return 0;
}
catch (CAltovaException& e)
```

Please note that the path names in the generated source code have been deleted for the sake of clarity.

Looking at **MappingMain1Object.Run**:
All parameters passed to the **run** method, **except for** the last one (CompletePO.xml), are source files. Please ensure that the last parameter is the target file.

In this example:

the **source** files are:

- XML file: Short.PO.xml
- Database file: CustomerAndArticles.mdb including the connection string

the **target** file is:

- CompletePO.xml

To define your own source or target files:

- Directly edit the parameters passed to the run method of MappingMain1Object

To use an XML input stream as the XML data source:

- Navigate to the run method declaration in the code, and configure the specific parameters there.

To add extra error handling code:

- Edit the code below the **`catch`** `(CAltovaException& e)` code.

22.6 Code generator options

The menu option **Tools | Options** lets you specify general as well as specific MapForce settings.

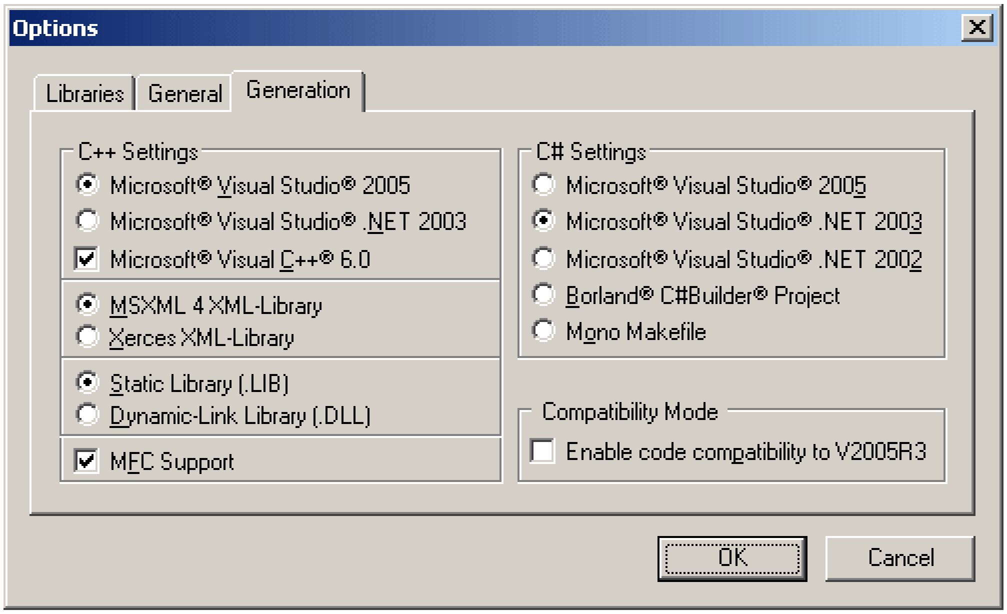
Generation tab:
This tab lets you define the settings for C++ and C# programming languages.

C++

- Microsoft Visual Studio 2005, .NET 2003 or Visual C++ 6.0 project file
- generate code using either MSXML 4 or Xerces XML library (note that MapForce currently supports Xerces Version 2.2 or higher)
- generate static libraries, or Dynamic-link libraries
- generate code with or without MFC support

C#
Select the type of project file you want to generate:

- Microsoft Visual studio 2005, 2003, or 2002 project file
- Borland C#Builder project
- Mono makefile



C++ Settings
- ● Microsoft® Visual Studio® 2005
- ○ Microsoft® Visual Studio® .NET 2003
- ☑ Microsoft® Visual C++® 6.0
- ● MSXML 4 XML-Library
- ○ Xerces XML-Library
- ● Static Library (.LIB)
- ○ Dynamic-Link Library (.DLL)
- ☑ MFC Support

C# Settings
- ○ Microsoft® Visual Studio® 2005
- ● Microsoft® Visual Studio® .NET 2003
- ○ Microsoft® Visual Studio® .NET 2002
- ○ Borland® C#Builder® Project
- ○ Mono Makefile

Compatibility Mode
- □ Enable code compatibility to V2005R3

Compatibility Mode:
Schema code (.xsd files) generated in versions after version V2005R3 is generated in a slightly different fashion to previous versions. This could lead to the situation that older code (from prior releases) and code generated by later releases, are not code compatible. For this reason a "compatibility mode" has been introduced, which generates code compatible to V2005R3.

- Compatibility Mode, Default is OFF.
- Please make it a point to use the new code generation features i.e., keep compatibility mode check box unchecked.
- The availability of the Compatibility Mode is only temporary, it will be removed in a future version.

Code generated code for XML schemas up till V2005R3 used a static DOM document instance as parent for all nodes.

```
{
    LibraryType lib = new LibraryType();
    BookType book = new BookType();
    book.addISBN( new SchemaString( "0764549642" ) );
    book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );

    LibraryDoc doc = new LibraryDoc();
    doc.setRootElementName( "http://www.nanonull.com/LibrarySample",
"Library" );
    doc.setSchemaLocation( "Library.xsd" ); // optional
    doc.save( "Library1.xml", lib );
}
```

Code generated for XML schemas after V2005R3:
The standard (non-parameter) constructor is not used, because it does not establish a reference to a DOM document. A new document is created first which can then be referenced by the root node. All new child nodes must then also be created in the correct document. This is accomplished by using the generated factory functions called newXXX, please see the example code below.

```
{
    LibraryDoc doc = new LibraryDoc();
    doc.setSchemaLocation( "Library.xsd" ); // optional

    // create root element with no namespace prefix
    LibraryType lib = new LibraryType(doc, "
http://www.nanonull.com/LibrarySample", "", "Library" );

    BookType book = lib.newBookType();    // factory functions are
generated for all members of a complex type
    book.addISBN( new SchemaString( "0764549642" ) );
    book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );

    doc.save( "Library1.xml", lib );
}
```

General tab:

- Specify if you want to show the logo on start and/or when printing.
- Enable/disable the MapForce gradient background
- Define the Default Output Encoding
- an execution timeout for the Output tab when previewing the mapping result.

Libraries tab:

- Add or delete user-defined XSLT, or programming language Libraries/functions to MapForce.

Mapping settings
Select the menu option **File | Mapping settings** to define:

- the **Application** name (name of generated Java files) used when generating code
- Base Package Name when compiling for Java.

22.7 The way to SPL (Spy Programming Language)

This section gives an overview of Spy Programming Language.

It is assumed that you have prior programming experience, and are familiar with operators, functions, variables and classes, as well as the basics of object-oriented programming - which is used heavily in SPL. You should also have detailed knowledge of XML Schema.

The templates used by MapForce are supplied in the **...**MapForce2006**\spl** folder. You can use these files as an aid to help you in developing your own templates.

How code generator works:
The basis of the generator are the template files (.spl). The template file is interpreted by the code generator and outputs a **.cpp, .java, .cs** source code file (or project), or any other type of file depending on the template. The source code is then compiled into an .exe file which can then be started. The .exe file then accesses the XML data described by the schema file.

SPL files have access to a wide variety of information that is collated from the source schemas. Please note that an SPL file is not tied to a specific schema, but allows access to all schemas! Make sure you write your SPL files generically, avoid structures etc. which apply to specific schemas!

Creating a new file in spl:

```
[create "test.cpp"]
#include "stdafx.h"
[close]
```

This is very basic SPL file. It creates a file named **test.cpp,** and places the include statement within it. The close command completes the template.

22.7.1 Code Blocks

Code generator instructions are enclosed in square brackets '**[**' and '**]**'.

Multiple instructions can be included in a bracket pair, additional instructions have to be separated by a new line or a colon '**:**'.

Valid examples are:

```
[$x = 0
$x = $x + 1]
```

or

```
[$x = 0: $x = $x +1]
```

22.7.2 Comments

Comments always begin with a **'** character, and terminate on the next line, or at a block close character **]**.

22.7.3 Variables

The **$** character is used when **declaring** or **using** a variable, a variable is always prefixed by **$**.

Variables types:

- integer
- string
- class iterator (see foreach statement)

Variable types are declared by first usage,

[$x = 0]
x is now an integer

[$x = "teststring"]
x is now treated as a string

Working with strings:
The previous example shows how a fixed string is assigned to a variable. You will also need to concatenate strings, as well as assign them to different variables.

Assigning a string to a variable:

[$x = $module]

Defining string concatenation uses the **&** character:

[$x = "testconcat" & "ination" & $module]

22.7.4 Global objects

After a Schema file is analyzed by the code generator, the objects in the table below exist in the Template Engine.

Name	Type	Description
$namespaces	Namespace collection	Collection of Namespace objects
$classes	Class collection	All the complextypes, elements,… in a flat view. The order is like the order while parsing through the types
$module	string	name of the source Schema without extension
$outputpath	string	The output path specified by the user, or the default output path

22.7.5 Using files

[create *filename*]
creates a new file - the file has to be closed with the **[close]** instruction.

[append *filename*]
appends items to a specific file - the file has to be closed with the **[close]** instruction.

22.7.6 Conditions

adding conditions to your code (if):
SPL allows you to use standard "if" instructions; the syntax is as follows:

```
[if  condition]
    [instruction block]
[else]
    [instruction block]
[endif]
```

or, **without else**:

```
[if  condition]
    [instruction block]
[endif]
```

Please note that there are no round brackets enclosing the condition!

Conditions:
As in any other programming language conditions are constructed with logical and comparison - operators which are:

Logical operators:

Operator	**Operator in SPL**
Not	not
And	and
Or	or

Comparison operators:

Not equal	<>
Equal	=
Greater than	>
Greater than or equal	>=
Less than	<
Less than or equal	<=

A sample **if** instruction with a condition looks like this:

```
[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
    whatever you want ['inserts whatever you want, in the resulting file]
[endif]
```

22.7.7 foreach

Vectors and iterators

A vector is a **collection** of objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.

foreach example:

```
[foreach $class in $classes]
    [if not $class.IsInternal]
      class [=$class.Name];
    [endif]
[next]
```

The first line:
$classes is the **global object** of all types. $classes is also a vector - a collection of single class objects.

Foreach steps through all the items in $classes, and executes the code within the instruction.

In each iteration, **$class** is assigned to the next class object. You simply work with the class object instead of using, classes[i]->class->Name(), as you would in C++.

foreach is executed till the **[next]** instruction.

22.7.8 Adding text to files

Iterating through collections and incrementing variables will not generate compilable files.

Text not enclosed by **[** and **]**, is written directly to the file. It's similar to using javascript in ASP.

If you want to place the value of a variable in the result document, use the following method:

> The result of your calculation is **[=$x]** - so have a nice day…
> - assuming that the value of variable x is 23, the file output will be:
> The result of your calculation is 23 - so have a nice day…

22.7.9 Subroutines

Code generator supports subroutines in the form of procedures or functions.

Features:

- By-value and by-reference passing of values
- Local/global parameters (local within subroutines)
- Recursive invocation (subroutines may call themselves)

Subroutine declaration

Subroutines

Syntax:

```
Sub SimpleSub()
... lines of code
EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

Please note:

Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters:

Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the **$** character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character "**,**" within round parentheses
- Parameters can pass values

Parameters - passing values

Parameters can be passed in two ways, by value and by reference, using the keywords ByVal and ByRef respectively.

Syntax:

```
' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] ...
```

- ByVal specifies that the parameter is passed by value.
- ByRef specifies that the parameter by reference.

Function return values

To return a value from a subroutine, use the **return** statement. Such a function can be called from within an expression.

Example:

```
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
  return $localName
else
  return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

Subroutine invocation

Use **call** to invoke a subroutine, followed by the procedure name and parameters, if any.

```
       Call SimpleSub()
```

or,

```
       Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

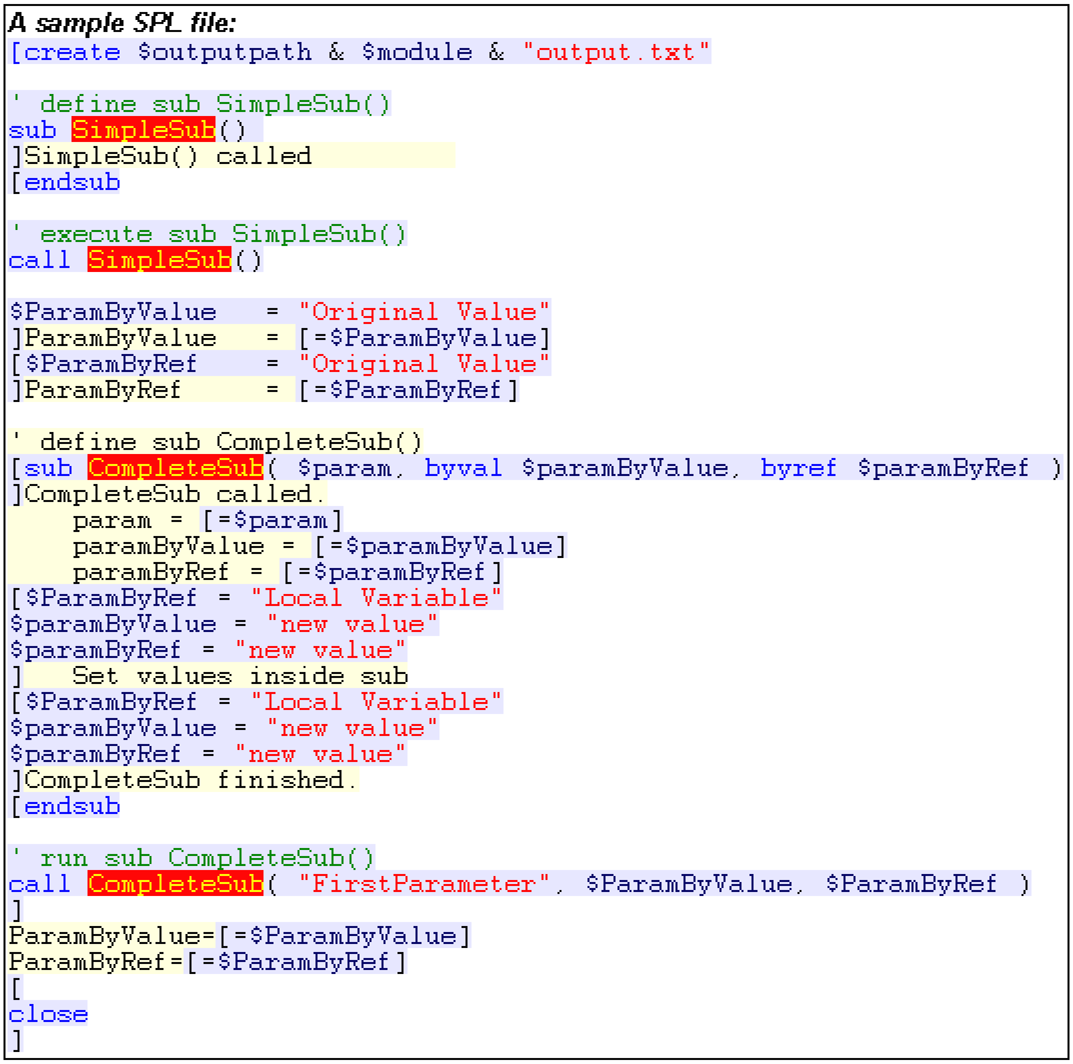
Legal parameter types:
Variables, strings, numbers, true and false, are the only data types that may currently be used within parameters. Do not use expressions!

To invoke a function (any subroutine that contains a **return** statement), simply use its name inside an expression. Do not use the **call** statement to call functions.
Example:

```
       $QName = MakeQualifiedName($namespace, "entry")
```

Subroutine example

Highlighted example showing subroutine declaration and invocation.



A sample SPL file:

```
[create $outputpath & $module & "output.txt"

' define sub SimpleSub()
sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
call SimpleSub()

$ParamByValue    = "Original Value"
]ParamByValue    = [=$ParamByValue]
[$ParamByRef     = "Original Value"
]ParamByRef      = [=$ParamByRef]

' define sub CompleteSub()
[sub CompleteSub( $param, byval $paramByValue, byref $paramByRef )
]CompleteSub called.
    param = [=$param]
    paramByValue = [=$paramByValue]
    paramByRef = [=$paramByRef]
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]   Set values inside sub
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]CompleteSub finished.
[endsub

' run sub CompleteSub()
call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[=$ParamByValue]
ParamByRef=[=$ParamByRef]
[
close
]
```

The same sample code:

```
[create $outputpath & $module & "output.txt"

' define sub SimpleSub()
Sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
Call SimpleSub()

$ParamByValue       = "Original Value"
]ParamByValue       = [=$ParamByValue]
[$ParamByRef = "Original Value"
]ParamByRef   = [=$ParamByRef]

' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
]CompleteSub called.
       param = [=$param]
       paramByValue = [=$paramByValue]
       paramByRef = [=$paramByRef]
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]      Set values inside Sub
[$ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]CompleteSub finished.
[endsub

' run sub CompleteSub()
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[=$ParamByValue]
ParamByRef=[=$ParamByRef]
[
Close
]
```

22.7.10 Built in Types

The section describes the built-in types used in global variables which describe the parsed schema.

Namespace

Namespace abstraction:

Name	Type	Description
URI	string	The URI of this namespace
Prefix	string	The prefix of this namespace
ContainsPublicClasses	boolean	True, if the Classes collection contains at least one non-internal Class object.
Classes	Class collection	Collection of the classes in this namespace - step through them using **foreach**

Class

Class abstraction:

Name	Type	Description
HasNamespace	boolean	True, if there is an associated Namespace object
Namespace	Namespace	The Namespace object this Class object is part of
NamespacePrefix	string	Prefix of the namespace in which the class is
NamespaceURI	string	URI of the namespace in which the class is
Name	string	Name of the type in the resulting file
SchemaName	string	Name of the type as in the original schema file
HasBaseObject	boolean	True, if this type is derived from another type, which is also represented by a Class object.
BaseObject	Class	The base Class object if HasBaseObject is true
BaseNamespaceURI	string	Namespace URI of the base class
Base	string	Name of the base class
SchemaBase	string	Name of the base type as in the original schema file
BuiltInBase	string	Name of the root simple type
BuiltInSchemaBase	string	Name of the root simple type as in the original schema file
IsRoot	boolean	True, if there is a root element of this type.
IsSimpleType	boolean	True, if this is a simple type.
IsComplexFromSimpleType	boolean	True, if this is a complex type and is derived from a simple type.
IsComplexType	boolean	True, if this is a complex type.
IsSequence	boolean	True, if the top-level group-type is "sequence".
IsChoice	boolean	True, if the top-level group-type is "choice".
IsAll	boolean	True, if the top-level group-type is "all".
Description	string	Description of this type. May contain line feeds.
IsGlobal	boolean	true if this type is usable anywhere in the generated files

IsAnonymous	boolean	true if this type is a kind of a help object that will not be shown in the generated files
IsInternal	boolean	true if this is a help object dealing with root elements
Members	Member collection	A class representing a complexType contains one or more Members.
Facets	Facet collection	A class representing a simpleType or a complexType derived from a simpleType may contain Facets.

Member

Abstraction of a member-variable inside a class.

Name	**Type**	**Description**
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
Name	string	Name in the resulting file
SchemaName	string	Name as in the original schema file
XmlName	string	The name as it is expected to appear in XML instance documents/streams.
Type	string	The name of the class which represents the schema type.
SchemaType	string	The schema type.
TypeObject	Class	See explanation below *****
HasTypeObject	boolean	true, when typeobject has a valid value
Description	string	Description of the Element/Attribute. Can contain line feeds.
IsBuiltInType	boolean	true, if the type is a built-in schema type, e.g. string, unsignedInt, dateTime...
IsSimpleType	boolean	true, if the type of this member is a simpleType
IsElement	boolean	true, if this is an element
IsAttribute	boolean	true, if this is an attribute
NodeType	string	"Element" or "Attribute"
MinOcc	integer	minOccurs, as in schema
MaxOcc	integer	maxOccurs, as in schema
IsQualified	boolean	true, if the form of the Element/Attribute is set to "qualified".

***TypeObject:**
The reason this has to be used is the following: you might use a complextype, and the complextype itself contains a simpletype.

- Get the $class variable, and step through its members using **foreach** - at some point you arrive at the simpletype.
- You now need the simpletype properties: type name, or whatever.
- Check the **HasTypeObject**, if its value is true, the member has been filled with valid contents and can be parsed.

Sample:

- For every type declared in the schema (xsd) a class is generated.
- All elements are generated as members inside the containing class.
- To find out which type the generated member has (which class we have to use to store the value) use the TypeObject-property.

Example:

```
<xs:complexType name="TypeA">
        <xs:sequence>
                <xs:element name="B" type="TypeB"/>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="TypeB"/>
```

CodeGen would create two classes:

```
class TypeA
{
 ….
 TypeB B;
}

class TypeB
{
 ….
}
```

Explanation:
When class "TypeA" is generated the member B is also generated. To find out the type of B, use the "TypeObject" method of the element which returns the class "TypeB". So we can get the name and use it there.

Facet

This class consists of the constraint itself, and several boolean variables.

The boolean variables tell you the kind of constraint you're currently dealing with. The names of the Boolean variables are identical to those used in the schema specification.

Name	Type	Description
Constraint	string	Holds the value of the facet
IsLength	boolean	
IsMinLength	boolean	
IsMaxLength	boolean	
IsMinInclusive	boolean	
IsMinExclusive	boolean	
IsMaxInclusive	boolean	
IsMaxExclusive	boolean	
IsTotalDigits	boolean	
IsFractionDigits	boolean	
IsWhiteSpace	boolean	
IsPattern	boolean	

IsEnumeration	boolean	
Enumeration	Enumeration collection	Holds a collection of Enumeration objects if this facet is of type enumeration.

Enumeration

Abstraction of an enumeration entry inside a facet.

Name	**Type**	**Description**
Index	integer	Holds the index of this enumeration value, starting with 1.
Value	string	Holds an enumeration value.

Chapter 23

The MapForce API

23 The MapForce API

The COM-based API of MapForce enables clients to easily access the functionality of MapForce. As a result, it is now possible to automate a wide range of tasks.

MapForce follows the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the MapForce API from common development environments, such as those using C, C++ and VisualBasic, and with scripting languages like JavaScript and VBScript.

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of MapForce may still remain in the system.
- See Error handling for details of how to avoid annoying error messages.
- Free references explicitly, if using languages such as C or C++.

23.1 Overview

This overview of the MapForce API provides you with the object model for the API and a description of the most important API concepts. The following topics are covered:

- The object model
- Example: Code-Generation
- Example: Project Support
- Error handling

23.1.1 Object model

The starting point for every application which uses the MapForce API is the `Application` object.

To create an instance of the `Application` object, call `CreateObject("MapForce.Application")` from VisualBasic, or a similar function from your preferred development environment, to create a COM object. There is no need to create any other objects to use the complete MapForce API. All other interfaces are accessed through other objects, with the `Application` object as the starting point.

The application object consists of the following parts (each indentation level indicates a child–parent relationship with the level directly above):

- `Application`
    - `Options`
    - `Project`
        - `ProjectItem`
    - `Documents`
        - `Document`
            - `MapForceView`
    - `ErrorMarkers`
        - `ErrorMarker`

Once you have created an `Application` object, you can start using the functionality of MapForce. You will generally either open an existing `Document`, create a new one, or generate code for or from this document.

23.1.2 Example: Code-Generation

See also

Code Generation

The following JScript example shows how to load an existing document and generate different kinds of mapping code for it.

```
// ------------------- begin JScript example ---------------------
// Generate Code for existing mapping.
// works with Windows scripting host.

// ----------------- helper function ------------------
function Exit(strErrorText)
{
       WScript.Echo(strErrorText);
       WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
       if (objErr != null)
              Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" +
objErr.description + " - " + strText);
       else
              Exit ("ERROR: " + strText);
}
// ---------------------------------------------------

// ----------------- MAIN ------------------

// ----- create the Shell and FileSystemObject of the windows scripting
try
{
       objWshShell = WScript.CreateObject("WScript.Shell");
       objFSO = WScript.CreateObject("Scripting.FileSystemObject");
}
catch(err)
       { Exit("Can't create WScript.Shell object"); }

// ----- open MapForce or access running instance and make it visible
try
{
       objMapForce = WScript.GetObject ("", "MapForce.Application");
       objMapForce.Visible = true;             // remove this line to perform
background processing
}
catch(err) { WScript.Echo ("Can't access or create MapForce.Application"); }

// ----- open an existing mapping. adapt this to your needs!
objMapForce.OpenDocument(objFSO.GetAbsolutePathName ("Test.mfd"));

// ----- access the mapping to have access to the code generation methods
var objDoc = objMapForce.ActiveDocument;

// ----- set the code generation output properties and call the code
generation methods.
// ----- adapt the output directories to your needs
try
{
       // ----- code generation uses some of these options
       var objOptions = objMapForce.Options;
       // ----- generate XSLT -----
       objOptions.XSLTDefaultOutputDirectory = "C:\\test\\TestCOMServer\\XSLT"
;
       objDoc.GenerateXSLT();

       // ----- generate Java Code -----
       objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\Java"
;
       objDoc.GenerateJavaCode();

       // ----- generate CPP Code, use same cpp code options as the last time
-----
       objOptions.CodeDefaultOutputDirectory = "C:\\test\\TestCOMServer\\CPP";
       objDoc.GenerateCppCode();

       // ----- generate C# Code, use options C# code options as the last time
-----
       objOptions.CodeDefaultOutputDirectory =
"C:\\test\\TestCOMServer\\CHash";
       objDoc.GenerateCHashCode();
}
catch (err)
       { ERROR ("while generating XSL or program code", err); }


// hide MapForce to allow it to shut down
objMapForce.Visible = false;

// -------------------- end example ---------------------
```

23.1.3 Example: Project Support

See also

Code Generation

The following JScript example shows how you can use the MapForce project and project-item objects of the MapForce API to automated complex tasks. Depending on your installation you might need to change the value of the variable `strSamplePath` to the example folder of your MapForce installation.

To successfully run all operations in this example below, you will need the Enterprise version of MapForce. If you have the Professional version running, you should comment out the lines that insert the WebService project. Users of the Standard edition will not have access to project-related functions at all.

```
// //////////// global variables /////////////////
var objMapForce = null;
var objWshShell = null;
var objFSO = null;

// !!! adapt the following path to your needs. !!!
var strSamplePath = "C:\\Program
Files\\Altova\\MapForce2006\\MapForceExamples\\";

// /////////////////////// Helpers //////////////////////////////

function Exit(strErrorText)
{
        WScript.Echo(strErrorText);
        WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
        if (objErr != null)
                Exit ("ERROR: (" + (objErr.number & 0xffff) + ")" +
objErr.description + " - " + strText);
        else
                Exit ("ERROR: " + strText);
}

function CreateGlobalObjects ()
{
        // the Shell and FileSystemObject of the windows scripting host often
always useful
        try
        {
                objWshShell = WScript.CreateObject("WScript.Shell");
                objFSO = WScript.CreateObject("Scripting.FileSystemObject");
        }
        catch(err)
                { Exit("Can't create WScript.Shell object"); }

        // create the MapForce connection
        // if there is a running instance of MapForce (that never had a
connection) - use it
        // otherwise, we automatically create a new instance
        try
        {
                objMapForce = WScript.GetObject("", "MapForce.Application");
        }
        catch(err)
        {
             { Exit("Can't access or create MapForce.Application"); }
       }
}

// ----------------------------------------------------------
// print project tree items and their properties recursively.
// ----------------------------------------------------------
function PrintProjectTree( objProjectItemIter, strTab )
{
       while ( ! objProjectItemIter.atEnd() )
       {
             // get current project item
             objItem = objProjectItemIter.item();

             try
             {
                    // ----- print common properties
                    strGlobalText += strTab + "[" + objItem.Kind + "]" +
objItem.Name + "\n";

                    // ----- print code generation properties, if available
                    try
                    {
                          if ( objItem.CodeGenSettings_UseDefault )
                                 strGlobalText += strTab + "  Use default
code generation settings\n";
                          else
                                 strGlobalText += strTab + "  code generation
language is " +

objItem.CodeGenSettings_Language +
                                                  " output folder is " +
objItem.CodeGenSettings_OutputFolder + "\n";
                    }
                    catch( err ) {}

                    // ----- print WSDL settings, if available
                    try
                    {
                          strGlobalText += strTab + "  WSDL File is " +
objItem.WSDLFile +
                                           " Qualified Name is " +
objItem.QualifiedName + "\n";
                    }
                    catch( err ) {}
             }
             catch( ex )
                    { strGlobalText += strTab + "[" + objItem.Kind + "]\n" }

             // ---- recurse
             PrintProjectTree( new Enumerator( objItem ), strTab + '  ' );

             objProjectItemIter.moveNext();
       }
}

// ----------------------------------------------------------
// Load example project installed with MapForce.
// ----------------------------------------------------------
function LoadSampleProject()
{
       // close open project
       objProject = objMapForce.ActiveProject;
       if ( objProject != null )
             objProject.Close();
       // open sample project and iterate through it.
       // sump properties of all project items

       objProject = objMapForce.OpenProject(strSamplePath +
"MapForceExamples.mfp");
       strGlobalText = '';
       PrintProjectTree( new Enumerator (objProject), ' ' )
       WScript.Echo( strGlobalText );

       objProject.Close();
}

// -------------------------------------------------------------
// Create a new project with some folders, mappings and a
// Web service project.
// -------------------------------------------------------------
function CreateNewProject()
{
       try
       {
             // create new project and specify file to store it.
             objProject = objMapForce.NewProject(strSamplePath + "Sample.mfp"
);

             // create a simple folder structure
             objProject.CreateFolder( "New Folder 1");
             objFolder1 = objProject.Item(0);
             objFolder1.CreateFolder( "New Folder 2");
             objFolder2 = ( new Enumerator( objFolder1 ) ).item();// an
alternative to Item(0)

             // add two different mappings to folder structure
             objFolder1.AddFile( strSamplePath + "DB_Altova_SQLXML.mfd");
             objMapForce.Documents.OpenDocument(strSamplePath +
"InspectionReport.mfd");
             objFolder2.AddActiveFile();

             // override code generation settings for this folder
             objFolder2.CodeGenSettings_UseDefault = false;
             objFolder2.CodeGenSettings_OutputFolder = strSamplePath +
"SampleOutput"
             objFolder2.CodeGenSettings_Language = 1;             //C++

             // insert Web service project based on a wsdl file from the
installed examples
             objProject.InsertWebService( strSamplePath +
"TimeService/TimeService.wsdl",

"{http://www.Nanonull.com/TimeService/}TimeService",
                                                            "TimeServiceSoap"
,
                                                            true );
             objProject.Save();
             if ( ! objProject.Saved )
                    WScript.Echo("problem occurred when saving project");

             // dump project tree
             strGlobalText = '';
             PrintProjectTree( new Enumerator (objProject), ' ' )
             WScript.Echo( strGlobalText );
       }
       catch (err)
       { ERROR("while creating new project", err ); }
}
```

```
// ----------------------------------------------------------
// Generate code for a project's sub-tree. Mix default code
// generation parameters and overloaded parameters.
// ----------------------------------------------------------
function GenerateCodeForNewProject()
{
      // since the Web service project contains only initial mappings,
      // we generate code only for our custom folder.
      // code generation parameters from project are used for Folder1,
      // whereas Folder2 provides overwritten values.
      objFolder = objProject.Item(0);
      objFolder1.GenerateCode();
}

// ////////////////////// MAIN //////////////////////////////

CreateGlobalObjects();
objMapForce.Visible = true;

LoadSampleProject();
CreateNewProject();
GenerateCodeForNewProject();

// uncomment to shut down application when script ends
// objMapForce.Visible = false;
```

23.1.4 Error handling

The MapForce API returns errors in two different ways. Every API method returns an HRESULT. This return value informs the caller about any malfunctions during the execution of the method. If the call was successful, the return value is equal to S_OK. C/C++ programmers generally use HRESULT to detect errors.

VisualBasic, scripting languages, and other high-level development environments do not give the programmer access to the returning HRESULT of a COM call. They use the second error-raising mechanism supported by the MapForce API, the IErrorInfo interface. If an error occurs, the API creates a new object that implements the IErrorInfo interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

The following text describes how to deal with errors raised from the MapForce API in different development environments.

VisualBasic
A common way to handle errors in VisualBasic is to define an error handler. This error handler can be set with the On Error statement. Usually the handler displays an error message and does some cleanup to avoid spare references and any kind of resource leaks. VisualBasic fills its own Err object with the information from the IErrorInfo interface.

Example:

```
Sub Validate()
  'place variable declarations here

  'set error handler
  On Error GoTo ErrorHandler

  'if generation fails, program execution continues at ErrorHandler:
  objMapForce.ActiveDocument.GenerateXSLT()

  'additional code comes here

  'exit
  Exit Sub

  ErrorHandler:
  MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
         "Description: " & Err.Description)
End Sub
```

JavaScript
The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, in that you also declare an error object containing the necessary information.

Example:

```
 Function Generate()
 {
  // please insert variable declarations here

  try
  {
   objMapForce.ActiveDocument.GenerateXSLT();
  }
  catch(Error)
  {
   sError = Error.description;
   nErrorCode = Error.number & 0xffff;
   return false;
  }

  return true;
 }
```

C/C++

C/C++ gives you easy access to the `HRESULT` of the COM call and to the `IErrorInterface`.

```
 HRESULT hr;

// Call GenerateXSLT() from the MapForce API
If(FAILED(hr = ipDocument->GenerateXSLT()))
{
  IErrorInfo *ipErrorInfo = Null;

  If(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
  {
   BSTRbstrDescr;
    ipErrorInfo->GetDescription(&bstrDescr);

   // handle Error information
      wprintf(L"Error message:\t%s\n",bstrDescr);
    ::SysFreeString(bstrDescr);

    // release Error info
    ipErrorInfo->Release();
  }
}
```

23.2 Object Reference

Object Hierarchy

```
Application
   Options
   Project
       ProjectItem
   Documents
      Document
         MapForceView
```

Enumerations

Description
This section contains the reference of the MapForce API 1.0 Type Library.

23.2.1 Application

Properties and Methods
Properties to navigate the object model:
Application
Parent
Options
Project
Documents

Application status:
Visible
Name
Quit
WindowHandle

MapForce designs:
NewDocument
OpenDocument
OpenURL
ActiveDocument

MapForce projects:
NewProject (Enterprise or Professional edition is required)
OpenProject (Enterprise or Professional edition is required)
ActiveProject (Enterprise or Professional edition is required)

MapForce code generation:
HighlightSerializedMarker

Examples
The following examples show how the automation interface of MapForce can be accessed from different programming environments in different languages.

```
' -------------------- begin VBA example ---------------------
' create a new instance of <SPY-MAP>.
Dim objMapForce As Application
Set objMapForce = CreateObject("MapForce.Application")
' --------------------- end example ---------------------

' -------------------- begin VBScript example ---------------------
' access a running, or create a new instance of MapForce.
' works with scripts running in the Windows scripting host.
Set objMapForce = GetObject("MapForce.Application");
' --------------------- end example ---------------------

// ------------------- begin JScript example ---------------------
// Access a running, or create a new instance of <MapForce
// works with scripts executed in the Windows scripting host
try
{
       objMapForce = WScript.GetObject ("", "MapForce.Application");
       // unhide application if it is a new instance
       objMapForce.Visible = true;
}
catch(err) { WScript.Echo ("Can't access or create MapForce.Application"); }
// -------------------- end example ---------------------
```

Events

This object supports the following events:

OnDocumentOpened
OnProjectOpened

OnDocumentOpened

Event: `OnDocumentOpened` (*`i_objDocument`* as `Document`)

Description
This event is triggered when an existing or new document is opened. The corresponding close event is `Document.OnDocumentClosed`.

OnProjectOpened

Event: `OnProjectOpened` (*`i_objProject`* as `Project`)

Description
This event is triggered when an existing or new project is loaded into the application. The corresponding close event is `Project.OnProjectClosed`.

ActiveDocument

Property: `ActiveDocument` as `Document` (read-only)

Description
Returns the automation object of the currently active document. This property returns the same as `Documents.ActiveDocument`.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

ActiveProject

Property: `ActiveProject` as `Project` (read-only)

Description
Returns the automation object of the currently active project.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1000	The application object is no longer valid.

1001 Invalid address for the return parameter was specified.

Documents

Property: `Documents` as [`Documents`]() (read-only)

Description
Returns a collection of all currently open documents.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

HighlightSerializedMarker

Method: `HighlightSerializedMarker` (*`i_strSerializedMarker`* as `String`)

Description
Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not already loaded, it will be loaded first. See [Document.GenerateCodeEx]() for a method to retrieve a serialized marker.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.
1007 The string passed in *i_strSerializedMarker* is not recognized a serialized MapForce marker.
1008 The marker points to a location that is no longer valid.

Name

Property: `Name` as `String` (read-only)

Description
The name of the application.

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

NewDocument

Method: `NewDocument` () as [`Document`]()

Description
Creates a new empty document. The newly opened document becomes the [`ActiveDocument`]() . This method is a shortened form of [`Documents.NewDocument`]().

Errors
1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

NewProject

Method: NewProject () as Project

Description
Creates a new empty project. The current project is closed. The new project is accessible under Project.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

OpenDocument

Method: OpenDocument (*i_strFileName* as String) as Document

Description
Loads a previously saved document file and continues working on it. The newly opened document becomes the ActiveDocument. This method is a shorter form of Documents.OpenDocument.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

OpenProject

Method: NewProject () as Project

Description
Opens an existing Mapforce project (*.mfp). The current project is closed. The newly opened project is accessible under Project.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

OpenURL

Method: OpenURL (*i_strURL* as String, *i_strUser* as String, *i_strPassword* as String)

Description
Loads a previously saved document file from an URL location. Allows user name and password to be supplied.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

Options

Property: Options as Options (read-only)

Description

This property gives access to options that configure the generation of code.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

Parent

Property: `Parent` as `Application` (read-only)

Description

The parent object according to the object model.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

Project

Property: `Project` as `Project` (read-only)

Description

Returns the MapForce project currently open. If no project is open, retuns `null`.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

Quit

Method: `Quit` ()

Description

Disconnects from MapForce to allow the application to shutdown. Calling this method is optional since MapForce keeps track of all external COM connections and automatically recognizes a disconnection. For more information on automatic shutdown see the `Visible` property.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.

Visible

Property: `Visible` as `Boolean`

Description

`True` if MapForce is displayed on the screen (though it might be covered by other applications or be iconized). `False` if MapForce is hidden. The default value for MapForce when automatically started due to a request from the automation server `MapForce.Application` is `false`. In all other cases, the property is initialized to `true`.

An application instance that is visible is said to be controlled by the user (and possibly by clients connected via the automation interface). It will only shut down due to an explicit user request.

To shut down an application instance, set its visibility to false and clear all references to this instance within your program. The application instance will shut down automatically when no further COM clients are holding references to it.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

WindowHandle

Property: `WindowHandle` () as long (read-only)

Description

Retrieve the application's Window Handle.

Errors

1000	The application object is no longer valid.
1001	Invalid address for the return parameter was specified.

23.2.2 MapForceView

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

View activation and view properties:
`Active`
`ShowItemTypes`
`ShowLibraryInFunctionHeader`
`HighlightMyConnections`
`HighlightMyConnectionsRecursivly`

Adding items:
`InsertXMLFile`
`InsertXMLSchema`
`InsertXMLSchemaWithSample`

Active

Property: `Active` as `Boolean`

Description
Use this property to query if the mapping view is the active view, or set this view to be the active one.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

HighlightMyConnections

Property: `HighlightMyConnections` as `Boolean`

Description
This property defines whether connections from the selected item only should be highlighed.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

HighlightMyConnectionsRecursivey

Property: `HighlightMyConnectionsRecursively` as `Boolean`

Description
This property defines if only the connections coming directly or indirectly from the selected item should be highlighed.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

InsertXMLFile

Method: `InsertXMLFile` (*`i_strXMLFileName`* as `String`, *`i_strRootElement`* as `String`)

Description
Adds a new item to the mapping. The item's internal structure is determined by the schema defined in the specified XML file. The second parameter defines the root element of this schema, if there is more than one candidate. The specified XML file is used as the input sample to evaluate the mapping.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

InsertXMLSchema

Method: `InsertXMLSchema` (*`i_strSchemaFileName`* as `String`, *`i_strRootElement`* as `String`)

Description
Adds a new item to the mapping. The item's internal structure is determined by the specified schema file. The second parameter defines the root element of this schema if there is more then one candidate. No XML input sample is assigned to this item.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

InsertXMLSchemaWithSample

Method: `InsertXMLSchemaWithSample` (*`i_strSchemaFileName`* as `String`, *`i_strXMLSampleName`* as `String`, *`i_strRootElement`* as `String`)

Description
Adds a new item to the mapping. The item's internal structure is determined by the specified schema file. The second parameter is stored as the XML input sample for mapping evaluation.The third parameter defines the root element of this schema if there is more then one candidate.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

Parent

Property: `Parent` as `Document` (read-only)

Description
The parent object according to the object model.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

ShowItemTypes

Property: `ShowItemTypes` as `Boolean`

Description
This property defines if types of items should be shown in the mapping diagram.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

ShowLibraryInFunctionHeader

Property: `ShowLibraryInFunctionHeader` as `Boolean`

Description
This property defines whether the name of the function library should be part of function names.

Errors

1300	The application object is no longer valid.
1301	Invalid address for the return parameter was specified.

23.2.3 Document

Properties and Methods

Properties to navigate the object model:
`Application`
`Parent`

File handling:
`FullName`
`Name`
`Path`
`Saved`
`Save`
`SaveAs`
`Close`

Code generation:
`OutputSettings_ApplicationName`
`OutputSettings_Encoding`
`JavaSettings_BasePackageName`

`GenerateXSLT`
`GenerateCppCode`
`GenerateJavaCode`
`GenerateCHashCode`
`GenerateCodeEx`
`HighlightSerializedMarker`

View access:
`MapForceView`

Events

This object supports the following events:

`OnDocumentClosed`
`OnModifiedFlagChanged`

OnDocumentClosed

Event: `OnDocumentClosed` (*`i_objDocument`* as `Document`)

Description
This event is triggered when a document is closed. The document object passed into the event handler should not be accessed. The corresponding open event is `Application.OnDocumentOpened`.

OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (*`i_bIsModified`* as Boolean)

Description
This event is triggered when a document's modification status changes.

Activate

Method: `Activate` ()

Description
Makes this document the active document.

Errors

1200	The application object is no longer valid.

Application

Property: `Application` as [`Application`]() (read-only)

Description
Retrieves the application's top-level object.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

Close

Method: `Close` ()

Description
Closes the document without saving.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

FullName

Property: `FullName` as String

Description
Path and name of the document file.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

GenerateCHashCode

Method: `GenerateCHashCode` ()

Description
Generate C# code that will perform the mapping. Uses the properties defined in `Application.Options` to configure code generation.

Errors

1200	The application object is no longer valid.

1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
`Code Generation`

GenerateCppCode

Method: `GenerateCppCode` ()

Description
Generates C++ code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
`Code Generation`

GenerateCodeEx

Method: `GenerateCodeEx (`*`i_nLanguage`*` as ENUMProgrammingLanguage) as ErrorMarkers`

Description
Generates C++ code that will perform the mapping. The parameter *i_nLanguage* specifies the target language. The method returns an object that can be used to enumerate all messages created the code generator. These are the same messages that get displayed in the Messages window of MapForce.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also
`Code Generation`

GenerateJavaCode

Method: `GenerateJavaCode` ()

Description
Generates Java code that will perform the mapping. Uses the properties defined in Application.Options to configure code generation.

Errors
1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1205 Error during code generation.

See also

`Code Generation`

GenerateOutput

Method: `GenerateOutput` ()

Description
Generates all output files defined in the mapping using a MapForce internal mapping language. The names of the output files are defined as properties of the output items in the mapping.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1206	Error during execution of mapping algorithm.

See also
`Code Generation`

GenerateXQuery

Method: `GenerateXQuery` ()

Description
Generates mapping code as XQuery. Uses the properties defined in Application.Options to configure code generation.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1204	Error during XSLT/XSLT2/XQuery code generation.

See also
`Code Generation`

GenerateXSLT

Method: `GenerateXSLT` ()

Description
Generates mapping code as XSLT. Uses the properties defined in Application.Options to configure code generation.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.
1204	Error during XSLT/XSLT2/XQuery code generation.

See also
`Code Generation`

GenerateXSLT2

Method: `GenerateXSLT2` ()

Description
Generates mapping code as XSLT2. Uses the properties defined in Application.Options to

configure code generation.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.
1204 Error during XSLT/XSLT2/XQuery code generation.

See also

`Code Generation`

HighlightSerializedMarker

Method: `HighlightSerializedMarker` (*`i_strSerializedMarker`* as `String`)

Description

Use this method to highlight a location in a mapping file that has been previously serialized. If the corresponding document is not already loaded, it will be loaded first. See GenerateCodeEx for a method to retrieve a serialized marker.

Errors

1000 The application object is no longer valid.
1001 Invalid address for the return parameter was specified.
1007 The string passed in *i_strSerializedMarker* is not recognized a serialized MapForce marker.
1008 The marker points to a location that is no longer valid.

JavaSettings_BasePackageName

Property: `JavaSettings_BasePackageName` as `String`

Description

Sets or retrieves the base package name used when generating Java code. This property is available in UI-dialog for the Document Settings.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

See also

`Code Generation`

MapForceView

Property: `MapForceView` as `Document` (read-only)

Description

This property gives access to functionality specific to the MapForce view.

Errors

1200 The application object is no longer valid.
1201 Invalid address for the return parameter was specified.

Name

Property: `Name` as `String`

Description
Name of the document file without file path.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

OutputSettings_ApplicationName

Property: `OutpuSettings_ApplicationName` as `String`

Description
Sets or retrieves the application name available in the Document Settings dialog.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

See also
`Code Generation`

OutputSettings_Encoding

Property: `OutputSettings_Encoding` as `String`

Description
Sets or retrieves the output encoding available in the Document Settings dialog.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

See also
`Code Generation`

Parent

Property: `Parent` as `Application` (read-only)

Description
The parent object according to the object model.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

Path

Property: `Path` as `String`

Description
Path of the document file without name.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

Save

Method: `Save` ()

Description
Save the document to the file defined by `Document.FullName`.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

SaveAs

Method: `SaveAs` (*`i_strFileName`* as `String`)

Description
Save document to specified file name, and set `Document.FullName` to this value if save operation was successful.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

Saved

Property: `Saved` as Boolean (read-only)

Description
`True` if the document was not modified since the last save operation, `false` otherwise.

Errors

1200	The application object is no longer valid.
1201	Invalid address for the return parameter was specified.

23.2.4 Documents

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Open and create mappings:
`OpenDocument`
`NewDocument`

Iterating through the collection:
`Count`
`Item`
`ActiveDocument`

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

Parent

Property: `Parent` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of documents in the collection.

Errors

1600	The object is no longer valid.
1601	Invalid address for the return parameter was specified.

Item

Property: `Item` (*`nIndex`* as `Integer`) as `Document` (read-only)

Description
Retrieves the document at `nIndex` from the collection. Indices start with 1.

Errors

1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

NewDocument

Method: `NewDocument` () as [`Document`]

Description
Creates a new document, adds it to the end of the collection, and makes it the active document.

Errors

1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

OpenDocument

Method: `OpenDocument` (*`strFilePath`* as `String`) as [`Document`]

Description
Opens an existing mapping document (`*.mfd`). Adds the newly opened document to the end of the collection and makes it the active document.

Errors

1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

ActiveDocument

Property: `ActiveDocument` as [Document] (read-only)

Description
Retrieves the active document. If no document is open, `null` is returned.

Errors

1600 The object is no longer valid.
1601 Invalid address for the return parameter was specified.

23.2.5 ErrorMarkers

Properties and Methods
Properties to navigate the object model:
Application
Parent

Iterating through the collection:
`Count`
`Item`

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves the number of error markers in the collection.

Errors

1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

Item

Property: `Item (`*`nIndex`*` as Integer) as ErrorMarker (read-only)`

Description
Retrieves the error marker at `nIndex` from the collection. Indices start with 1.

Errors

1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

Parent

Property: `Parent` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1800	The object is no longer valid.
1801	Invalid address for the return parameter was specified.

23.2.6 ErrorMarker

Properties and Methods
Properties to navigate the object model:
Application
Parent

Access to message information:

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

DocumentFileName

Property: `DocumentFileName` as `String` (read-only)

Description
Retrieves the name of the mapping file that the error marker is associated with.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

ErrorLevel

Property: `ErrorLevel` as `ENUMCodeGenErrorLevel` (read-only)

Description
Retrieves the severity of the error.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

Highlight

Method: `Highlight()`

Description
Highlights the item that the error marker is associated with. If the corresponding document is not open, it will be opened.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.
1008	The marker points to a location that is no longer valid.

Serialization

Property: `Serialization` as `String` (read-only)

Description
Serialize error marker into a string. Use this string in calls to Application.HighlightSerializedMarker or Document.HighlightSerializedMarker to highlight the marked item in the mapping. The string can be persisted and used in other instantiations of MapForce or its Control.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

Text

Property: `Text` as `String` (read-only)

Description
Retrieves the message text.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

Parent

Property: `Parent` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1900	The object is no longer valid.
1901	Invalid address for the return parameter was specified.

23.2.7 Options

This object gives access to all MapForce options available in the **Tools | Options** dialog.

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

General options:
`ShowLogoOnPrint`
`ShowLogoOnStartup`
`UseGradientBackground`

Options for code generation:
`CompatibilityMode`
`DefaultOutputEncoding`
`XSLTDefaultOutputDirectory`
`CodeDefaultOutputDirectory`
`CppSettings_DOMType`
`CppSettings_LibraryType`
`CppSettings_UseMFC`
`CppSettings_GenerateVC6ProjectFile`
`CppSettings_GenerateVSProjectFile`
`CSharpSettings_ProjectType`

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the application's top-level object.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

CodeDefaultOutputDirectory

Property: `CodeDefaultOutputDirectory` as `String`

Description
Specifies the target directory where files generated by `Document.GenerateCppCode`, `Document.GenerateJavaCode` and `Document.GenerateCHashCode`, are placed.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

See also
`Code Generation`

CompatibilityMode

Property: `CompatibilityMode` as `Boolean`

Description

Set to true to generate code compatible with Version 2005R3. Set to false to use newly added code generation features in `Document.GenerateCppCode`, `Document.GenerateCHashCode`, `Document.GenerateJavaCode` and `Document.GenerateXSLT`

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also

`Code Generation`

CppSettings_DOMType

Property: `CppSettings_DOMType` as `ENUMDOMType`

Description

Specifies the DOM type used by `Document.GenerateCppCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also

`Code Generation`

CppSettings_GenerateVC6ProjectFile

Property: `CppSettings_GenerateVC6ProjectFile` as `Boolean`

Description

Specifies if VisualC++ 6.0 project files should be generated by `Document.GenerateCppCode`.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also

`Code Generation`

CppSettings_GenerateVSProjectFile

Property: `CSharpSettings_GenerateVSProjectFile` as `ENUMProjectType`

Description

Specifies which version of VisualStudio project files should be generated by `Document.GenerateCppCode`.
Only `eVisualStudio2003Project` and `eVisualStudio2005Project` are valid selections.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also
`Code Generation`

CppSettings_LibraryType

Property: `CppSettings_LibraryType` as `ENUMLibType`

Description
Specifies the library type used by `Document.GenerateCppCode`.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

See also
`Code Generation`

CppSettings_UseMFC

Property: `CppSettings_UseMFC` as `Boolean`

Description
Specifies if MFC support should be used by C++ code generated by `Document.GenerateCppCode`.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

See also
`Code Generation`

CSharpSettings_ProjectType

Property: `CSharpSettings_ProjectType` as `ENUMProjectType`

Description
Specifies the type of C# project used by `Document.GenerateCHashCode`.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

See also
`Code Generation`

DefaultOutputEncoding

Property: `DefaultOutputEncoding` as `Boolean`

Description
File encoding used for output files.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

See also

`Code Generation`

Parent

Property: `Parent` as Application (read-only)

Description

The parent object according to the object model.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

ShowLogoOnPrint

Property: `ShowLogoOnPrint` as `Boolean`

Description

Show or hide the MapForce logo on printed outputs.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

ShowLogoOnStartup

Property: `ShowLogoOnStartup` as `Boolean`

Description

Show or hide the MapForce logo on application startup.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

UseGradientBackground

Property: `UseGradientBackground` as `Boolean`

Description

Set or retrieve the background color mode for a mapping window.

Errors

1400 The application object is no longer valid.
1401 Invalid address for the return parameter was specified.

XSLTDefaultOutputDirectory

Property: `XSLTDefaultOutputDirectory` as `String`

Description
Specifies the target directory where files generated by `Document.GenerateXSLT` are placed.

Errors

1400	The application object is no longer valid.
1401	Invalid address for the return parameter was specified.

See also
`Code Generation`

23.2.8 Project (Enterprise or Professional Edition)

Properties and Methods

Properties to navigate the object model:
Application
Parent

File handling:
FullName
Name
Path
Saved
Save
Close

Project tree navigation:
Count
Item
_NewEnum

Project tree manipulation:
AddActiveFile
AddFile
InsertWebService (Enterprise edition only)
CreateFolder

Code-generation:
Output_Folder
Output_Language
Output_TextEncoding
Java_BasePackageName
GenerateCode
GenerateCodeEx
GenerateCodeIn
GenerateCodeInEx

For examples of how to use the properties and methods listed above, see Example: Project Support. Note that, in order to use these properties and methods, you will need to have the Enterprise or Professional edition of MapForce installed on your computer. For operations with Web services, the Enterprise edition is required.

_NewEnum

Property: _NewEnum () as IUnknown (read-only)

Description

This property supports language-specific standard enumeration.

Errors

1500	The object is no longer valid.

Examples

```
// ---------------------------------------------------------
// JScript sample - enumeration of a project's project items.
function AllChildrenOfProjectRoot()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {
        for ( objProjectIter = new Enumerator(objProject); !
objProjectIter.atEnd(); objProjectIter.moveNext() )
        {
            objProjectItem = objProjectIter.item();

            // do something with project item here
        }
    }
}

// ------------------------------------------------------------
// JScript sample - iterate all project items, depth first.
function IterateProjectItemsRec(objProjectItemIter)
{
    while ( ! objProjectItemIter.atEnd() )
    {
        objProjectItem = objProjectItemIter.item();
        // do something with project item here

        IterateProjectItemsRec( new Enumerator(objProjectItem) );

        objProjectItemIter.moveNext();
    }
}
function IterateAllProjectItems()
{
    objProject = objMapForce.ActiveProject;
    if ( objProject != null )
    {
        IterateProjectItemsRec( new Enumerator(objProject) );
    }
}
```

Events

This object supports the following events:

OnProjectClosed

OnProjectClosed

Event: OnProjectClosed (*i_obj*Project as Project)

Description
This event is triggered when the project is closed. The project object passed into the event handler should not be accessed. The corresponding open event is Application.OnProjectOpened.

AddActiveFile

Method: AddActiveFile () as ProjectItem

Description
Adds the currently open document to the mapping folder of the project's root.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

1503 No active document is available.
1504 Active documents needs to be given a path name before it can be added to the project.
1705 Mapping could not be assigned to project. Maybe it is already contained in the target folder.

AddFile

Method: `AddFile` (*`i_strFileName`* as `String`) as [`ProjectItem`]

Description
Adds the specified document to the mapping folder of the project's root.

Errors
1500 The object is no longer valid.
1501 The file name is empty.
Invalid address for the return parameter was specified.
1705 Mapping could not be assigned to project.
The file does not exist or is not a MapForce mapping.
Maybe the file is already assigned to the target folder.

Application

Property: `Application` as [`Application`] (read-only)

Description
Retrieves the top-level application object.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

Close

Method: `Close` ()

Description
Closes the project without saving.

Errors
1500 The object is no longer valid.

Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves number of children of the project's root item.

Errors
1500 The object is no longer valid.

Examples
See [`Item`] or [`_NewEnum`].

CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as [`ProjectItem`]

Description
Creates a new folder as a child of the project's root item.

Errors

1500	The object is no longer valid.
1501	Invalid folder name or invalid address for the return parameter was specified.

FullName

Property: `FullName` as String (read-only)

Description
Path and name of the project file.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

GenerateCode

Method: `GenerateCode` ()

Description
Generates code for all project items of the project. The code language and output location is determined by properties of the project and project items.

Errors

1500	The object is no longer valid.
1706	Error during code generation

GenerateCodeEx

Method: `GenerateCode` () `as` [`ErrorMarkers`]

Description
Generates code for all project items of the project. The code language and output location are determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.
1706	Error during code generation

GenerateCodeIn

Method: `GenerateCodeIn` (*`i_nLanguage`* as [`ENUMProgrammingLanguage`])

Description
Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items.

Errors

1500	The object is no longer valid.
1706	Error during code generation

GenerateCodeInEx

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`) as `ErrorMarkers`

Description
Generates code for all project items of the project in the specified language. The output location is determined by properties of the project and project items. An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.
1706	Error during code generation

InsertWebService

Method: `InsertWebService` (*`i_strWSDLFile`* as `String`, *`i_strService`* as `String`, *`i_strPort`* as `String`, *`i_bGenerateMappings`* as `Boolean)` as `ProjectItem`

Description
Inserts a new Web service project into the project's Web service folder. If `i_bGenerateMappings` is true, initial mapping documents for all ports get generated automatically.

Errors

1500	The object is no longer valid.
1501	WSDL file can not be found or is invalid. Service or port names are invalid. Invalid address for the return parameter was specified.
1503	Operation not supported by current edition.

Item

Property: `Item(`*`i_nItemIndex`* as `Integer)` as `ProjectItem` (read-only)

Description
Returns the child at `i_nItemIndex` position of the project's root. The index is zero-based. The largest valid index is `Count-1`. For an alternative to visit all children see `_NewEnum`.

Errors

1500	The object is no longer valid.

Examples

```
// ----------------------------------------------------------
// JScript code snippet - enumerate children using Count and Item.
for( nItemIndex = 0; nItemIndex < objProject.Count; nItemIndex++ )
{
    objProjectItem = objProject.Item(nItemIndex);
    // do something with project item here
}
```

Java_BasePackageName

Property: Java_BasePackageName as String

Description
Sets or gets the base package name of the Java packages that will be generated. This property is used only when generating Java code.

Errors

1500	The object is no longer valid.
1501	Invalid package name specified. Invalid address for the return parameter was specified.

Name

Property: Name as String (read-only)

Description
Name of the project file without file path.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

Output_Folder

Property: Output_Folder as String

Description
Sets or gets the default output folder used with GenerateCode and GenerateCodeIn. Project items can overwrite this value in their CodeGenSettings_OutputFolder property, when CodeGenSettings_UseDefault is set to false.

Errors

1500	The object is no longer valid.
1501	Invalid folder name specified. Invalid address for the return parameter was specified.

Output_Language

Property: OutputLanguage as ENUMProgrammingLanguage

Description
Sets or gets the default language for code generation when using GenerateCode. Project items can overwrite this value in their CodeGenSettings_OutputLanguage property, when CodeGenSettings_UseDefault is set to false.

Errors

1500 The object is no longer valid.
1501 Invalid language specified.
Invalid address for the return parameter was specified.

Output_TextEncoding

Property: `Output_TextEncoding` as `String`

Description
Sets or gets the text encoding used when generating XML-based code.

Errors
1500 The object is no longer valid.
1501 Invalid text encoding specified.
Invalid address for the return parameter was specified.

Parent

Property: `Parent` as `Application` (read-only)

Description
Retrieves the top-level application object.

Errors
1500 The is no longer valid.
1501 Invalid address for the return parameter was specified.

Path

Property: `Path` as String (read-only)

Description
Path of the project file without name.

Errors
1500 The object is no longer valid.
1501 Invalid address for the return parameter was specified.

Save

Method: `Save` ()

Description
Saves the project to the file defined by `FullName`.

Errors
1500 The object is no longer valid.
1502 Can't save to file.

Saved

Property: `Saved` as Boolean (read-only)

Description

`True` if the project was not modified since the last Save operation, `false` otherwise.

Errors

1500	The object is no longer valid.
1501	Invalid address for the return parameter was specified.

23.2.9 ProjectItem (Enterprise or Professional Edition)

Properties and Methods
Properties to navigate the object model:
`Application`
`Parent`

Project tree navigation:
`Count`
`Item`
`_NewEnum`

Project item properties:
`Kind`
`Name`
`WSDLFile` (only available to Web service project items)
`QualifiedName` (only available to Web service project items)

Project tree manipulation:
`AddActiveFile` (only available to folder items)
`AddFile` (only available to folder items)
`CreateFolder` (only available to folder items)
`CreateMappingForProject` (only available to Web service operations)
`Remove`

Document access:
`Open` (only available to mapping items and Web service operations)

Code-generation:
`CodeGenSettings_UseDefault`
`CodeGenSettings_OutputFolder`
`CodeGenSettings_Language`
`GenerateCode`
`GenerateCodeEx`
`GenerateCodeIn`
`GenerateCodeInEx`

For examples of how to use the properties and methods listed above, see Example: Project Support. Note that, in order to use these properties and methods, you will need to have the Enterprise or Professional edition of MapForce installed on your computer. For operations with Web services, the Enterprise edition is required.

_NewEnum

Property: `_NewEnum` () as `IUnknown` (read-only)

Description
This property supports language specific standard enumeration.

Errors

1700	The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

AddActiveFile

Method: `AddActiveFile` () as `ProjectItem`

Description
Adds the currently active document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

Errors

1700	The object is no longer valid.
1701	The file name is empty. Invalid address for the return parameter was specified.
1703	No active document is available.
1704	Active documents needs to be given a path name before it can be added to the project.
1705	Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.

AddFile

Method: `AddFile` (*`i_strFileName`* as `String`) as `ProjectItem`

Description
Adds the specified document to this project item if it is a valid child. Otherwise, the document is added to the Mapping Folder of the project's root.

Errors

1700	The object is no longer valid.
1701	The file name is empty. Invalid address for the return parameter was specified.
1705	Mapping could not be assigned to project. The file does not exist or is not a MapForce mapping. Maybe the file is already assigned to the target folder.

Application

Property: `Application` as `Application` (read-only)

Description
Retrieves the top-level application object.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

CodeGenSettings_Language

Property: `CodeGenSettings_Language` as `ENUMProgrammingLanguage`

Description
Gets or sets the language to be used with `GenerateCode` or `Project.GenerateCode`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.

Errors

1700 The object is no longer valid.
1701 Invalid language or invalid address for the return parameter was specified.

CodeGenSettings_OutputFolder

Property: `CodeGenSettings_OutputFolder` as `String`

Description
Gets or sets the output directory to be used with `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` or `Project.GenerateCodeIn`. This property is consulted only if `CodeGenSettings_UseDefault` is set to false.

Errors
1700 The object is no longer valid.
1701 An invalid output folder or an invalid address for the return parameter was specified.

CodeGenSettings_UseDefault

Property: `CodeGenSettings_UseDefault` as `Boolean`

Description
Gets or sets whether output directory and code language are used as defined by either (a) the parent folders, or (b) the project root. This property is used with calls to `GenerateCode`, `GenerateCodeIn`, `Project.GenerateCode` and Project.GenerateCodeIn. If this property is set to false, the values of `CodeGenSettings_OutputFolder` and `CodeGenSettings_Language` are used to generate code for this project item..

Errors
1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.

Count

Property: `Count` as `Integer` (read-only)

Description
Retrieves number of children of this project item. Also see `Item`.

Errors
1700 The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

CreateFolder

Method: `CreateFolder` (*`i_strFolderName`* as `String`) as `ProjectItem`

Description
Creates a new folder as a child of this project item.

Errors

1700 The object is no longer valid.
1701 Invalid folder name or invalid address for the return parameter was specified.
1702 The project item does not support children.

CreateMappingForProject

Method: `CreateMappingForProject` (*`i_strFileName`* as `String`) as [`ProjectItem`]

Description
Creates an initial mapping document for a Web service operation and saves it to `i_strFileName`. When using [`Project.InsertWebService`] you can use the `i_bGenerateMappings` flag to let MapForce automatically generate initial mappings for all ports.

Errors
1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.
1707 Cannot create new mapping.
The project item does not support auto-creation of initial mappings or a mapping already exists.
1708 Operation not supported in current edition.

GenerateCode

Method: `GenerateCode` ()

Description
Generates code for this project item and its children. The code language and output location is determined by [`CodeGenSettings_UseDefault`], [`CodeGenSettings_Language`] and [`CodeGenSettings_OutputFolder`]. Children of this project item can have their own property settings related to code-generation.

Errors
1700 The object is no longer valid.
1706 Error during code generation.

GenerateCodeEx

Method: `GenerateCode` () `as` [`ErrorMarkers`]

Description
Generates code for this project item and its children. The code language and output location are determined by [`CodeGenSettings_UseDefault`], [`CodeGenSettings_Language`] and [`CodeGenSettings_OutputFolder`]. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors
1700 The object is no longer valid.
1701 Invalid address for the return parameter was specified.
1706 Error during code generation.

GenerateCodeIn

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`)

Description
Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

Errors

1700 The object is no longer valid.
1701 Invalid language specified.
1706 Error during code generation.

GenerateCodeInEx

Method: `GenerateCodeIn` (*`i_nLanguage`* as `ENUMProgrammingLanguage`) `as ErrorMarkers`

Description
Generates code for the project item and its children in the specified language. The output location is determined by `CodeGenSettings_UseDefault` and `CodeGenSettings_OutputFolder`. Children of this project item can have their own property settings related to code-generation.

An object that can be used to iterate through all messages issued by the code generation process is returned. These messages are the same as those shown in the *Messages* window of MapForce.

Errors

1700 The object is no longer valid.
1701 Invalid language specified or invalid address for the return parameter was specified.
1706 Error during code generation.

Item

Property: `Item(`*`i_nItemIndex`* as `Integer)` as `ProjectItem` (read-only)

Description
Returns the child at `i_nItemIndex` position of this project item. The index is zero-based. The largest valid index is `Count` - 1.
For an alternative to visit all children see `_NewEnum`.

Errors

1700 The object is no longer valid.

Examples
See `Project.Item` or `Project._NewEnum`.

Kind

Property: `Kind` as `ENUMProjectItemType` (read-only)

Description

Retrieves the kind of the project item. Availability of some properties and the applicability of certain methods is restricted to specific kinds of project items. The description of all methods and properties contains information about these restrictions.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

Name

Property: `Name` as `String`

Description
Retrieves or sets the name of a project item. The name of most items is read-only. Exceptions are user-created folders, the names of which can be altered after creation.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	Project item does not allow to alter its name.

Open

Method: `Open` () as Document

Description
Opens the project item as a document or makes the corresponding document the active one, if it is already open. The project item must be a MapForce mapping or, for Enterprise edition only, Web service operation.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item does not refer to a MapForce mapping file.
1708	Operation not supported in current edition.

Parent

Property: `Parent` as Project (read-only)

Description
Retrieves the project that this item is a child of. Has the same effect as `Application.ActiveProject`.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.

QualifiedName

Property: `QualifiedName` as `String` (read-only)

Description
Retrieves the qualified name of a Web service item.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item is not a part of a Web service.

Remove

Method: `Remove` ()

Description

Remove this project item and all its children from the project tree.

Errors

1700	The object is no longer valid.

WSDLFile

Property: `WSDLFile` as `String` (read-only)

Description

Retrieves the file name of the WSDL file defining the Web service that hosts the current project item.

Errors

1700	The object is no longer valid.
1701	Invalid address for the return parameter was specified.
1702	The project item is not a part of a Web service.

23.3 Enumerations

This is a list of all enumerations used by the MapForce API. If your scripting environment does not support enumerations, use the number-values instead.

23.3.1 ENUMCodeGenErrorLevel

Description
Enumeration values to identify severity of code generation messages.

Possible values:

eCodeGenErrorLevel_Information	= 0
eCodeGenErrorLevel_Warning	= 1
eCodeGenErrorLevel_Error	= 2
eCodeGenErrorLevel_Undefined	= 3

23.3.2 ENUMDOMType

Description
Enumeration values to specify the DOM type used by generated C++ mapping code.

Possible values:

eDOMType_msxml4	= 0
eDOMType_xerces	= 1

See also
`Code Generation`

23.3.3 ENUMLibType

Description

Enumeration values to specify the library type used by the generated C++ mapping code.

Possible values:

eLibType_static	= 0
eLibType_dll	= 1

See also

`Code Generation`

23.3.4 ENUMProgrammingLanguage

Description
Enumeration values to select a programming language.

Possible values:

eUndefinedLanguage	= -1
eJava	= 0
eCpp	= 1
eCSharp	= 2
eXSLT	= 3
eXSLT2	= 4
eXQuery	= 5

23.3.5 ENUMProjectItemType

WDescription
Enumeration the different kinds of project items that can be children of Project or folder-like ProjectItems.

Possible values:

eProjectItemType_Invalid	= -1
eProjectItemType_MappingFolder	= 0
eProjectItemType_Mapping	= 1
eProjectItemType_WebServiceFolder	= 2
eProjectItemType_WebServiceRoot	= 3
eProjectItemType_WebServiceService	= 4
eProjectItemType_WebServicePort	= 5
eProjectItemType_WebServiceOperation	= 6
eProjectItemType_ExternalFolder	= 7
eProjectItemType_LibrarzFolder	= 8
eProjectItemType_ResourceFolder	= 9
eProjectItemType_VirtualFolder	= 10

See also
`ProjectItem.Kind`

23.3.6 ENUMProjectType

Description

Enumeration values to select a project type for generated C# mapping code.

Possible values:

eVisualStudioProject	= 0
eVisualStudio2003Project	= 1
eBorlandProject	= 2
eMonoMakefile	= 3
eVisualStudio2005Project	= 4

See also

`Code Generation`

23.3.7 ENUMViewMode

Description
Enumeration values to select a MapForce view.

Possible values:

eMapForceView	= 0
eXSLView	= 1
eOutputView	= 2

Chapter 24

MapForceControl

24 MapForceControl

MapForceControl is a control that provides a means of integration of the MapForce user interface and the functionality described in this section into most kinds of applications. ActiveX technology was chosen so as to allow integration using any of a wide variety of languages; this enables C++, C#, VisualBasic, or HTML to be used for integration. All components are full OLE Controls, which makes integration as simple as possible. Two different levels of integration are provided, thus enabling the integration to be adapted to a wide range of needs.

For a successful integration you have to consider the following main design factors:

- What technology or programming language can the hosting application use to integrate the MapForceControl?
- Should the integrated UI look exactly like MapForce with all its menus, toolbars, and windows, or will a subset of these elements—like allowing only one document and a restricted set of commands—be more effective?
- How deep will the integration be? Should the MapForce user interface be used as is? Are user interface extensions and/or restrictions required? Can some frequently used tasks be automated?

The sections, Integration at the Application Level and Integration at Document Level, both of which have examples in various programming languages, will help you to make the right decisions quickly. The section, Object Reference, describes all COM objects that can be used for integration, together with their properties and methods.

For automation tasks, the MapForce Automation Interface is accessible from the MapForceControl as well.

24.1 Integration at the Application Level

Integration at application level is simple and straightforward. It allows you to embed the complete interface of MapForce into a window of your application. Since you get the whole user interface of MapForce, you get all menus, toolbars, the status bar, document windows, and helper windows. Customization of the application's user interface is restricted to what MapForce provides. This includes rearrangement and resizing of helper windows and customization of menus and toolbars.

The only ActiveX control you need to integrate is `MapForceControl`. Its property `IntegrationLevel` defaults to application-level. You may use `Appearance` and `BorderStyle` to configure the appearance of the control's wrapper window. Do not instantiate or access `MapForceControlDocument` or `MapForceControlPlaceHolder` ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of MapForce, use the properties, methods, and events described for `MapForceControl`. Consider using `MapForceControl.Application` for more complex access to MapForce functionality.

In this section is an example (Example: HTML) showing how the MapForce application can be embedded in an HTML page. For usage with other programming languages, or more sophisticated access, see the Examples of integration at document-level.

24.1.1 Example: HTML

This example shows a simple integration of the MapForce control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a MapForceControl in HTML code.
- Implement buttons to load documents and automate code-generation tasks.
- Define actions for some application events.

The code for this example is available at the following location in your MapForce installation: `MapForceExamples\ActiveX\HTML\MapForceActiveX_ApplicationLevel.htm`.

Instantiate the Control

The HTML `Object` tag is used to create an instance of the MapForceControl. The `Classid` is that of MapForceControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```
<OBJECT id="objMapForceControl"
        Classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
        width="800"
        height="500"
        VIEWASTEXT>
</OBJECT>
```

Add Button to Open Default Document

As a simple example of how to automate some tasks, we add a button to the page:

```
<input type="button" value="Open Marketing Expenses"
onclick="BtnOpenMEFile()">
```

When clicked, a predefined document will be opened in the MapForceControl. We use a method to locate the file relative to the MapForceControl so the example can run on different installations.

```
<SCRIPT ID=Javahandlers LANGUAGE=javascript>
// ---------------------------------
// open a pre-defined document
function BtnOpenMEFile()
{
 var pos = objMapForceControl.BaseHref.indexOf("ActiveX");

 if(pos > 7)
 {
  path = objMapForceControl.BaseHref.substr(7, pos - 7);          // remove
file protocol

  objMapForceControl.Open(path + "MarketingExpenses.mfd");
 }
 else
 {
  alert("Unable to locate MarketingExpenses.mfd at: " +
objMapForceControl.BaseHref);
 }
}
</SCRIPT>
```

Add Buttons for Code Generation

Although code-generation for the active document is available via menus, we want to have buttons that will generate code without asking the user for the location of the output. The method is similar to that used in the previous section.

First come the buttons:

```
<input type="button" value="Generate XSLT" onclick="BtnGenerate( 0 )">
<input type="button" value="Generate Java" onclick="BtnGenerate( 1 )">
<input type="button" value="Generate C++" onclick="BtnGenerate( 2 )">
<input type="button" value="Generate C#" onclick="BtnGenerate( 3 )">
```

Then we provide the script that will generate the code into sub-folders of the currently defined default output folders.

```
<SCRIPT ID=Javahandlers LANGUAGE=javascript>
// ---------------------------------------------------------------------
// generate code for active document into language-specific sub folders of
// the current default output directory. No user interaction necessary.
function BtnGenerate(languageID)
{
       // get top-level object of automation interface
       var objApp = objMapForceControl.Application;

       // get the active document
       var objDocument = objApp.ActiveDocument;

       // retrieve object to set the generation output path
       var objOptions = objApp.Options;

       if ( objDocument == null )
              alert( "no active document found" );
       else
       {
              if (languageID == 0)
              {
                     objOptions.XSLTDefaultOutputDirectory =
objOptions.XSLTDefaultOutputDirectory + "\\XSLTGen";
                     objDocument .GenerateXSLT();
              }
              else if (languageID == 1)
              {
                     objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/JavaCode";
                     objDocument .GenerateJavaCode();
              }
              else if (languageID == 2)
              {
                     objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/CPPCode";
                     objDocument .GenerateCppCode();
              }
              else if (languageID == 3)
              {
                     objOptions.CodeDefaultOutputDirectory =
objOptions.CodeDefaultOutputDirectory + "/CSharpCode";
                     objDocument .GenerateCHashCode();
              }
       }
}
</SCRIPT>
```

Connect to Custom Events

The example implements two event callbacks for MapForceControl custom events to show the principle:

```
<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentOpened" of MapForceControl object  -->
<SCRIPT FOR="objMapForceControl" event="OnDocumentOpened( objDocument )"
LANGUAGE="javascript">
      // alert("Document '" + objDocument.Name + "' opened!");
</SCRIPT>

<!-- ----------------------------------------------------------- -->
<!--  custom event 'OnDocumentClosed" of MapForceControl object  -->
<SCRIPT FOR="objMapForceControl" event="OnDocumentClosed( objDocument )"
LANGUAGE="javascript">
      // alert("Document '" + objDocument.Name + "' closed!");
</SCRIPT>
```

24.2 Integration at Document Level

Integration at document level gives you freedom over instantiation and placement of the following parts of the MapForce user interface:

- Editing windows for MapForce mappings
- MapForce overview window
- MapForce library window
- MapForce validation window
- MapForce project window

If necessary, a replacement for the menus and toolbars of MapForce must be provided by your application.

You will need to instantiate and access multiple ActiveX controls, depending on which user interface parts you want to re-use. All these controls are contained in the MapForceControl OCX.

- Use MapForceControl to set the integration level and access application wide functionality.
- Use MapForceControlDocument to create any number of editor windows. It may be sufficient to create only one window and re-use it for different mapping files, depending on your needs.
- Optionally Use MapForceControlPlaceholder to embed MapForce overview, library, validation and project windows.
- Access run-time information about commands, menus, and toolbars available in MapForceControl to seamlessly integrate these commands into your application's menus and toolbars. See Use MapForceCommands for more information.

If you want to automate some behaviour of MapForce use the properties, methods, and events described for the MapForceControl, MapForceControlDocument and MapForceControlPlaceHolder. Consider using MapForceControl.Application, MapForceControlDocument.Document and MapForceControlPlaceHolder.Project for more complex access to MapForce functionality. However, to open a document always use MapForceControlDocument.OpenDocument or MapForceControlDocument.NewDocument on the appropriate document control. To open a project always use MapForceControlPlaceHolder.OpenProject on a placeholder control embedding a MapForce project window.

See Examples on how to instantiate and access the necessary controls in different programming environments.

24.2.1 Use MapForceControl

To integrate at document level, instantiate a `MapForceControl` first. Set the property `IntegrationLevel` to `ICActiveXIntegrationOnDocumentLevel(=1)`. Set the window size of the embedding window to `0x0` to hide any user interface behind the control. You may use `Appearance` and `BorderStyle` to configure the appearance of the control's wrapper window.

Avoid using the method `Open` since this might lead to unexpected results. Use the corresponding open methods of `MapForceControlDocument` and `MapForceControlPlaceHolder`, instead.

See Query MapForce Commands for a description of how to integrate MapForce commands into your application. Send commands to MapForce via the method `Exec`. Query if a command is currently enabled or disabled using the method `QueryStatus`.

24.2.2 Use MapForceControlDocument

An instance of the `MapForceControlDocument` ActiveX control allows you to embed one MapForce mapping editing window into your application. You can use any number of instances you need. Each instance will have one mapping loaded. New instances contain a new mapping at creation. Use the method `OpenDocument` to load any other existing mapping file.

The control supports a read-only mode via the property `ReadOnly`.

Use `Path` and `SaveDocument` or methods and properties accessible via the property `Document` to access document functionality.

24.2.3 Use MapForceControlPlaceHolder

Instances of MapForceControlPlaceHolder ActiveX controls allow you to selectively embed the additional helper windows of MapForce into your application. The property `PlaceholderWindowID` selects the MapForce helper window to be embedded. Use only one MapForceControlPlaceHolder for each window identifier. See `Enumerations.MapForceControlPlaceholderWindow` for valid window identifiers.

For placeholder controls that select the MapForce project window, additional methods are available. Use `OpenProject` to load a MapForce project. Use the property `Project` and the methods and properties from the MapForce automation interface to perform any other project related operations.

24.2.4 Query MapForce Commands

When integrating at document-level, no menu or toolbar from MapForce is available to your application. Instead, you can query all the commands and the structure of the application menu at runtime. Use the property `MapForceControl.CommandsStructure` to access this information. Professional applications will need to integrate this menu in a sophisticated manner into their own menu structure. Your installation of MapForce even provides you with command label images used within MapForce. See the folder `MapForceExamples\ActiveX\Images` of your MapForce installation for icons in GIF format. The file names correspond to the `labels` of commands.

See the C# Example for details of how to use the command structure information to create a menu at runtime.

24.2.5 Examples

This section contains examples of MapForce document-level integration using different container environments and programming languages. Source code for all examples is available in the folder `MapForceExamples\ActiveX` of your MapForce installation.

C#

The C# example shows how to integrate the MapForceControl in a common desktop application created with C# using Visual Studio .NET 2003. The following topics are covered:

- Building a dynamic menu bar based on information the MapForceControl API provides.
- Usage of MapForce Placeholder controls in a standard frame window.
- Usage of a MapForce Placeholder control in a sizeable Tool Window.
- How to handle an event raised by the MapForceControl API.

Please note that the example application is already complete. There is no need to change anything if you want to run and see it working. The following steps describe what general actions and considerations must be taken in order to create a project such as this.

Introduction

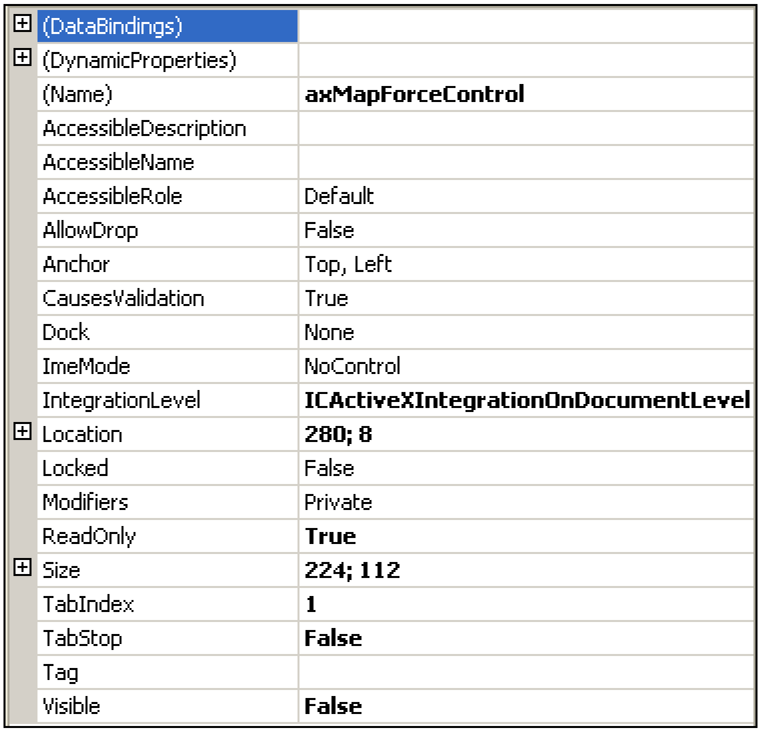
Adding the MapForce components to the Toolbox
Before you take a look at the sample project please add the assemblies to the .NET IDE Toolbox. The MapForce Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC). If you open the Toolbox dialog under **Tools | Add/Remove Toolbox Items** the controls will appear as `AxMapForceControl`, `AxMapForceControlDocument` and `AxMapForceControlPlaceholder` on the .NET Framework Components tab. Check all to make them available to the IDE.

Now you can open the `MapForceApplication.sln` file in the `ActiveX\C#\MapForceApplication` folder to load the project.

Placing the MapForceControl

It is necessary to have one MapForceControl instance to set the integration level and to manage the Document and Placeholder controls of the MapForce library. The control is accessible via the General section of the Toolbox helper window in the IDE. To add it you need to select the component in the Toolbox window and drag a rectangle wherever you want to have it in the destination window. If you have an application which does not open a window on startup you can use a simple invisible Form with the control on it which is created manually in the code.

The example project adds this instance to the main MdiContainer MDIMain. If you open MDIMain in the Design View from the Solution Explorer you will see a light blue rectangle at the top-left side in the client area of the Frame window. Selecting this rectangle will show you the properties of the MapForceControl. It is important to set the `IntegrationLevel` property to **`ICActiveXIntegrationOnDocumentLevel`** in order to turn on the Document and Placeholder support of the MapForce library. Properties of the MapForceControl component placed in the MDIFrame Window of the example application are shown below:



⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControl**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	None
ImeMode	NoControl
IntegrationLevel	**ICActiveXIntegrationOnDocumentLevel**
⊞ Location	**280; 8**
Locked	False
Modifiers	Private
ReadOnly	**True**
⊞ Size	**224; 112**
TabIndex	**1**
TabStop	**False**
Tag	
Visible	**False**

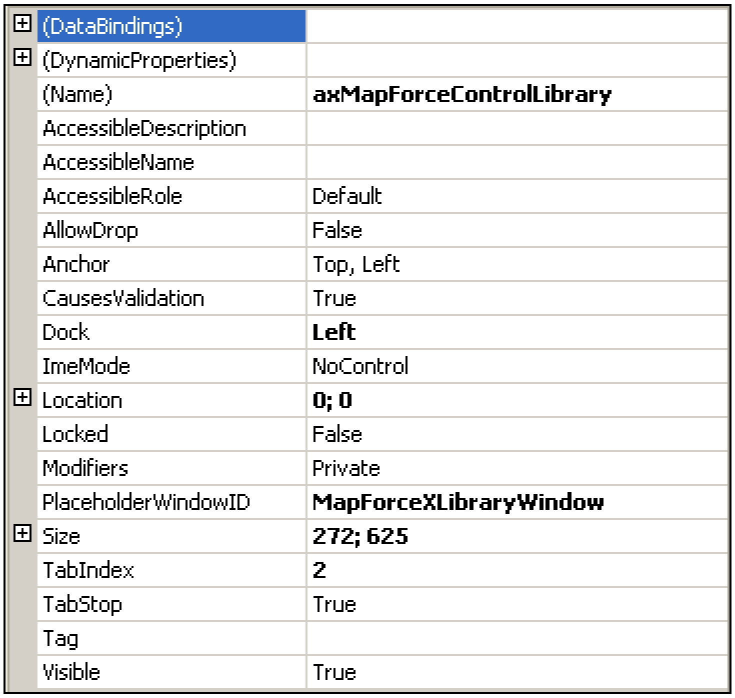
Set the Visible flag to False to avoid any confusion about the control for the user.

Adding the Placeholder Controls

Placeholders on the MDI Frame

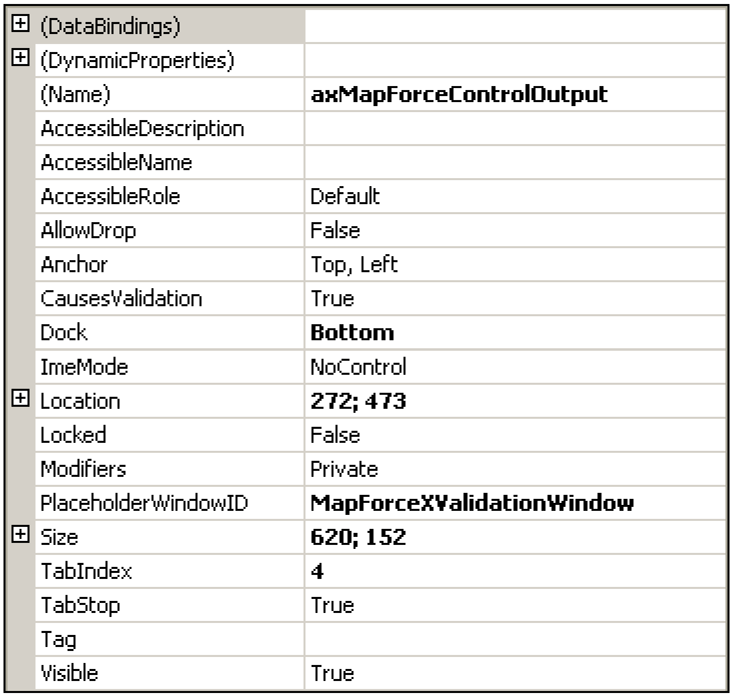
The example project has to place Placeholder controls on the main MDI Frame. They are also added via the Toolbox window by dragging a rectangle on the destination Form. To set the type of the Placeholder which should be displayed one has to set the `PlaceholderWindowID` property. This property can also be changed during runtime in the code of the application. The Placeholder control would change its content immediately.

Properties of the Library window on the left side of the MDIMain Frame window are shown below:



⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControlLibrary**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	**Left**
ImeMode	NoControl
⊞ Location	**0; 0**
Locked	False
Modifiers	Private
PlaceholderWindowID	**MapForceXLibraryWindow**
⊞ Size	**272; 625**
TabIndex	**2**
TabStop	True
Tag	
Visible	True

Properties of the Output window at the bottom:



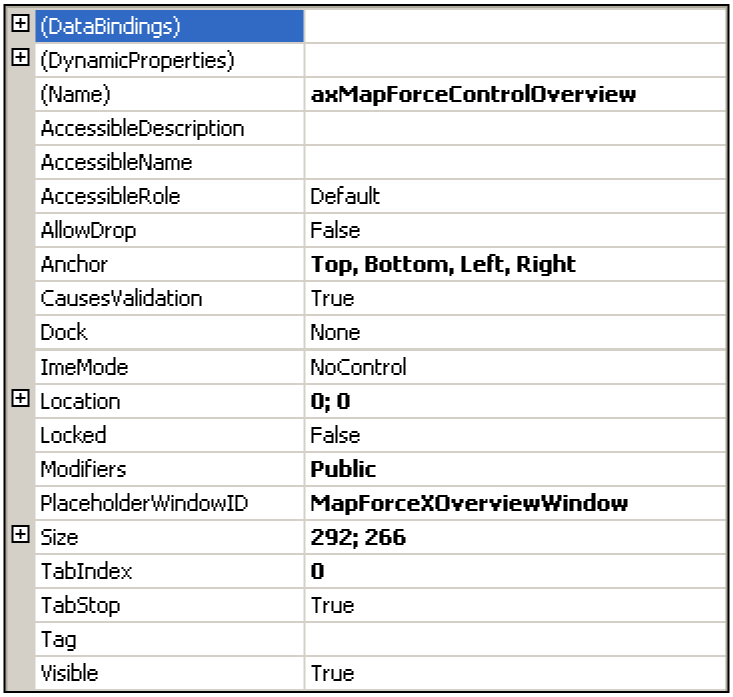
⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControlOutput**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
CausesValidation	True
Dock	**Bottom**
ImeMode	NoControl
⊞ Location	**272; 473**
Locked	False
Modifiers	Private
PlaceholderWindowID	**MapForceXValidationWindow**
⊞ Size	**620; 152**
TabIndex	**4**
TabStop	True
Tag	
Visible	True

The Placeholders also have the Anchor and Dock properties set in order to react on resizing of the Frame window.

Placeholder on a separate Toolwindow
It is also possible to place a Placeholder control on a separate floating Toolwindow. To do this, create a new Form as a Toolwindow and add the control as shown above. The

MapForceOverviewWnd in the sample project contains the Overview window of MapForce.

Properties of the Overview Toolwindow:



⊞ (DataBindings)	
⊞ (DynamicProperties)	
(Name)	**axMapForceControlOverview**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	**Top, Bottom, Left, Right**
CausesValidation	True
Dock	None
ImeMode	NoControl
⊞ Location	**0; 0**
Locked	False
Modifiers	**Public**
PlaceholderWindowID	**MapForceXOverviewWindow**
⊞ Size	**292; 266**
TabIndex	**0**
TabStop	True
Tag	
Visible	True

However, all Placeholder controls need a connection to the main MapForceControl. Normally this connection can be established automatically and there is nothing more to do. The two placeholders on the MDI Frame work like this. In the case of the Placeholder control in the Toolwindow, we need to add some code to the `public MDIMain()` method in `MDIMain.cs`:

```
m_MapForceOverview = new MapForceOverviewWnd();

MapForceControlLib.MapForceControlPlaceHolderClass type =
(MapForceControlLib.MapForceControlPlaceHolderClass)m_MapForceOverview.axM
apForceControlOverview.GetOcx();
type.AssignMultiDocCtrl((MapForceControlLib.MapForceControlClass)axMapForc
eControl.GetOcx());

m_MapForceOverview.Show();
```

The MapForceOverviewWnd is created and shown here. In addition, a special method of the Placeholder control is called in order to connect the MapForcecontrol to it. `AssignMultiDocCtrl()` takes the MapForceControl as parameter and registers a reference to it in the Placeholder control.

Retrieving Command Information

The MapForceControl gives access to all commands of MapForce through its `CommandsStructure` property. The example project uses the `MapForceCommands` and `MapForceCommand` interfaces to dynamically build a menu in the MDI Frame window which contains most of the MapForce commands.

The code to add the commands is placed in the `MDIMain` method of the `MapForceApplication` class in the file `MDIMain.cs`:

```
public MDIMain()
```

```
{
    .
    .
    .
    MFLib.MapForceCommands objCommands;
    objCommands = axMapForceControl.CommandsStructure;

    long nCount = objCommands.Count;

    for(long idx = 0;idx < nCount;idx++)
    {
        MFLib.MapForceCommandobjCommand;
        objCommand = objCommands[(int)idx];

        // We are looking for the Menu with the name IDR_MAPFORCE. This menu
        contains
        // the complete main menu of MapForce.

        if(objCommand.Label == "IDR_MAPFORCE")
        {
            InsertMenuStructure(mainMenu.MenuItems, 1, objCommand, 0, 0,
            false);
        }
    }
    .
    .
    .
}
```

`mainMenu` is the name of the menu object of the MDI Frame window created in the Visual Studio IDE. `InsertMenuStructure` takes the MapForce menu from the IDR_MAPFORCE command object and adds the MapForce menu structure to the already existing menu of the sample project. No commands from the **File**, **Project**, or **Window** menu are added.

The new commands are instances of the class `CustomMenuItem`, which is defined in `CustomMenuItem.cs`. This class has an additional member to save the MapForce command ID, which is taken to execute the command using Exec on selecting the menu item. This code from `InsertMenuStructure` creates the new command:

```
CustomMenuItem   newMenuItem = new CustomMenuItem();

if(objCommand.IsSeparator)
    newMenuItem.Text = "-";
else
{
    newMenuItem.Text = strLabel;
    newMenuItem.m_MapForceCmdID = (int)objCommand.ID;
    newMenuItem.Click += new EventHandler(AltovaMenuItem_Click);
}
```

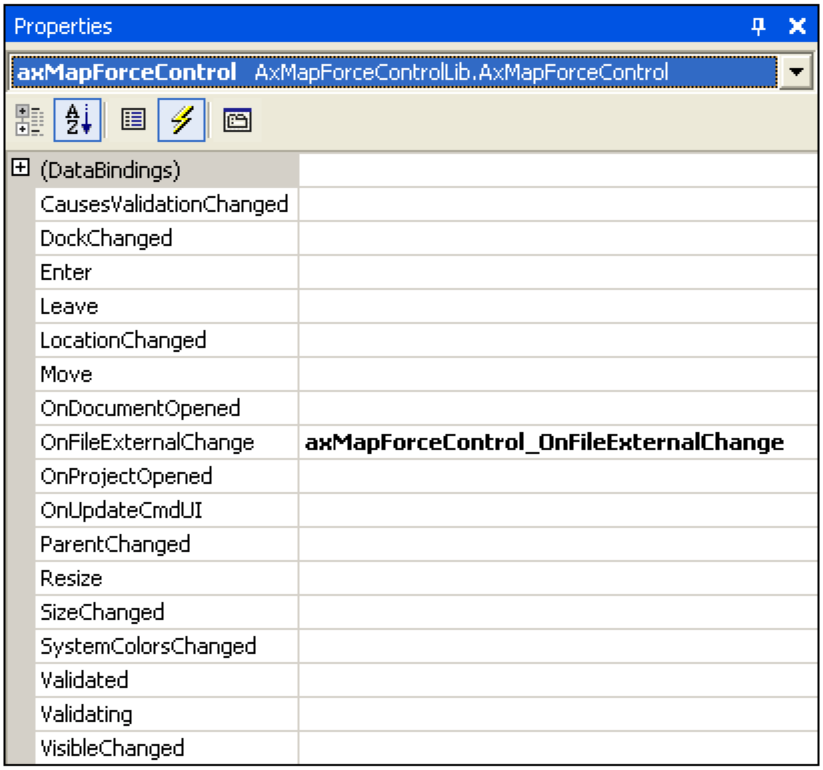
You can see that all commands get the same event handler AltovaMenuItem_Click which does the processing of the command:

```
private void AltovaMenuItem_Click(object sender, EventArgs e)
{
    if(sender.GetType() ==
    System.Type.GetType("MapForceApplication.CustomMenuItem"))
    {
        CustomMenuItemcustomItem = (CustomMenuItem)sender;

        ProcessCommand(customItem.m_MapForceCmdID);
    }
}

private void ProcessCommand(int nID)
{
    MapForceDoc docMapForce = GetCurrentMapForceDoc();

    if(docMapForce != null)
        docMapForce.axMapForceControlDoc.Exec(nID);
    else
        axMapForceControl.Exec(nID);
}
```

`ProcessCommand` delegates the execution either to the MapForceControl itself or to any active MapForce document loaded in a `MapForceControlDocument` control. This is necessary because the MapForceControl has no way to know which document is currently active in the hosting application.

Handling Events

Because all events in the MapForce library are based on connection points, you can use the C# delegate mechanism to provide the custom event handlers. You will always find a complete list of events on the property page of each control of the MapForce library. The picture below shows the events of the main MapForceControl:



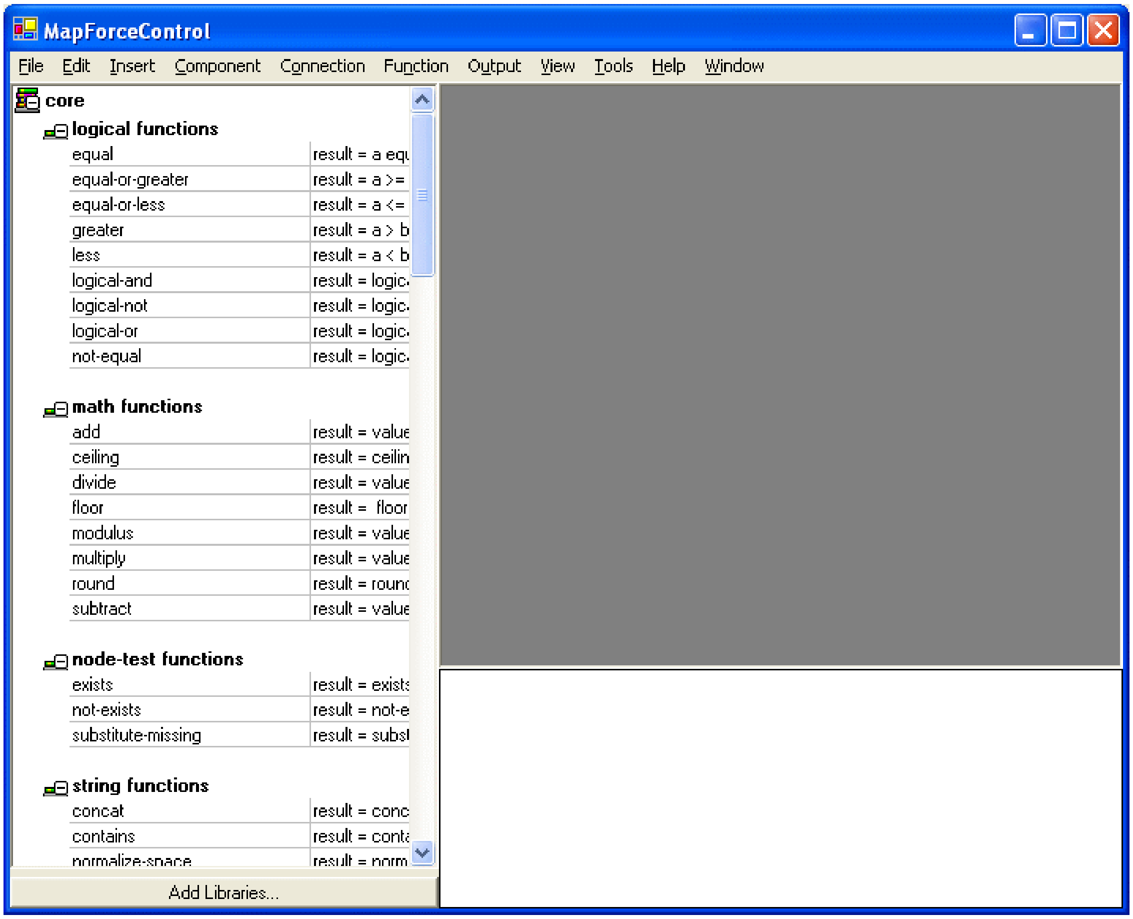
(DataBindings)	
CausesValidationChanged	
DockChanged	
Enter	
Leave	
LocationChanged	
Move	
OnDocumentOpened	
OnFileExternalChange	**axMapForceControl_OnFileExternalChange**
OnProjectOpened	
OnUpdateCmdUI	
ParentChanged	
Resize	
SizeChanged	
SystemColorsChanged	
Validated	
Validating	
VisibleChanged	

As you can see, the example project only overrides the `OnFileExternalChange` event. The creation of the C# delegate is done for you by the C# Framework. All you need to do is to fill the empty event handler. The handler implementation turns off any file reloading and displays a message box to inform the user that a file loaded by the MapForceControl has been changed from outside:

```
private void axMapForceControl_OnFileExternalChange(object sender,
AxMapForceControlLib._DMapForceControlEvents_OnFileExternalChangeEvent e)
{
    MessageBox.Show("Attention: The file " + e.strPath + " has been changed
from outside\nbut reloading is turned off in the sample application!");

    // This turns off any file reloading:
    e.varRet = false;
}
```

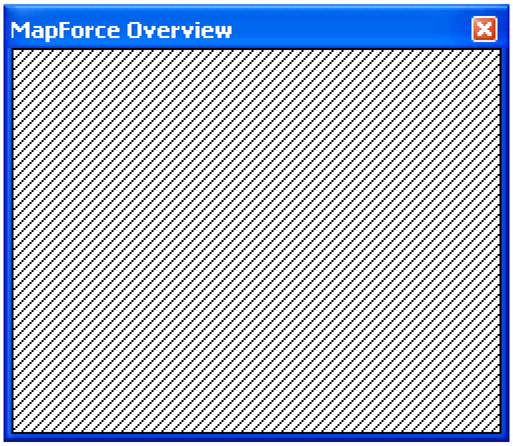
Testing the Example

After adding the assemblies to the Toolbox (see Introduction), you can run the sample project with F5 without the need to change anything in the code. The main MDI Frame window is created together with a floating Toolwindow containing the Overview window of MapForce. The application looks something like the screenshot below:

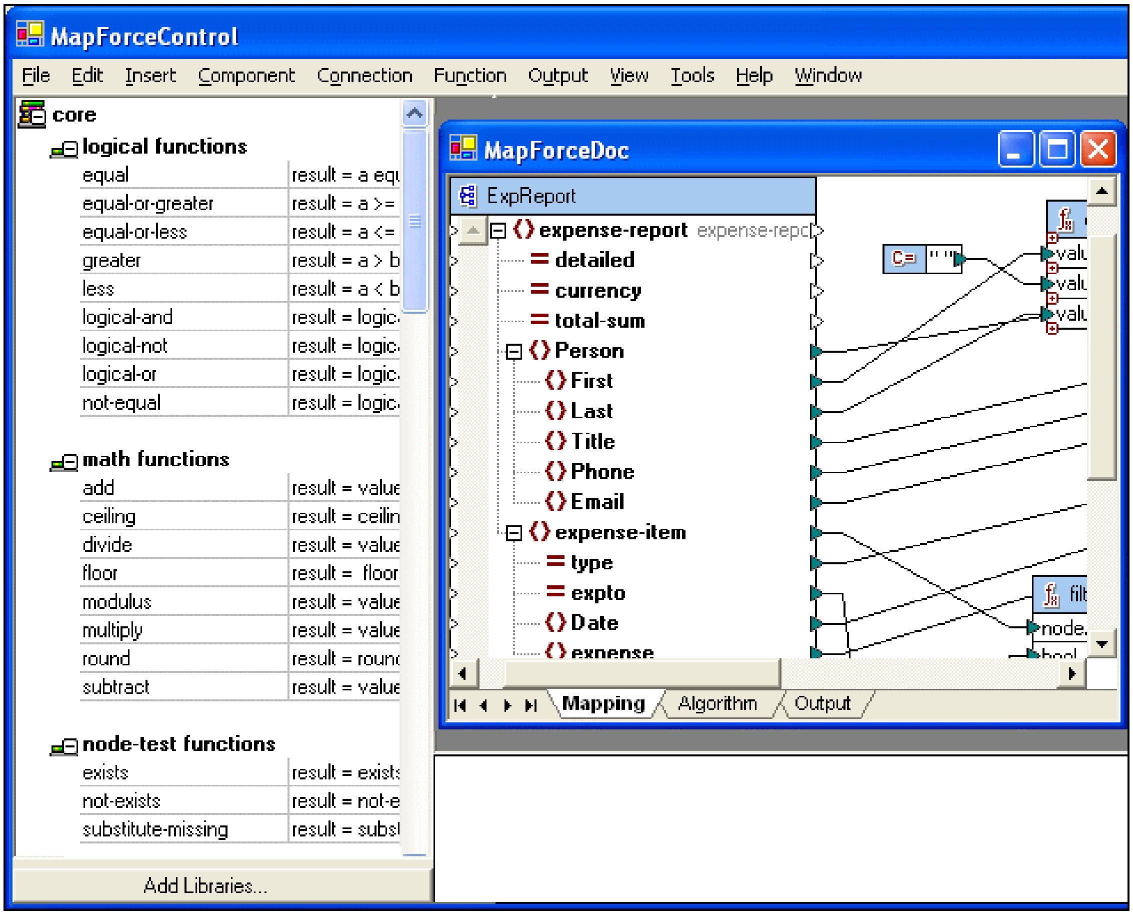

MapForceControl
File Edit Insert Component Connection Function Output View Tools Help Window
core
Add Libraries...

logical functions	
equal	result = a equ
equal-or-greater	result = a >=
equal-or-less	result = a <=
greater	result = a > b
less	result = a < b
logical-and	result = logic
logical-not	result = logic
logical-or	result = logic
not-equal	result = logic
math functions	
add	result = value
ceiling	result = ceilin
divide	result = value
floor	result = floor
modulus	result = value
multiply	result = value
round	result = round
subtract	result = value
node-test functions	
exists	result = exists
not-exists	result = not-e
substitute-missing	result = subst
string functions	
concat	result = conc
contains	result = conta
normalize-space	result = norm

The floating Overview Toolwindow is also created:



Use **File | Open** to open the file `MarketingExpenses.mfd`, which is in the MapForce examples folder. The file is loaded and displayed in an own document child window:



equal	result = a equ
equal-or-greater	result = a >=
equal-or-less	result = a <=
greater	result = a > b
less	result = a < b
logical-and	result = logic
logical-not	result = logic
logical-or	result = logic
not-equal	result = logic

add	result = value
ceiling	result = ceilin
divide	result = value
floor	result = floor
modulus	result = value
multiply	result = value
round	result = rounc
subtract	result = value

exists	result = exists
not-exists	result = not-e
substitute-missing	result = subst

After you load the document, you can try using menu commands. Note that context menus are also available. If you like, you can also load additional documents. Save any modifications using the **File | Save** command.

HTML

This example shows an integration of the MapForce control at document-level into a HTML page. The following topics are covered:

- Instantiate a MapForceControl ActiveX control object in HTML code
- Instantiate a MapForceControlDocument ActiveX control to allow editing a MapForce mapping
- Instantiate one MapForceControlPlaceHolder for a MapForce project window
- Instantiate one MapForceControlPlaceHolder ActiveX control to alternatively host one of the MapForce helper windows
- Create a customer toolbar for some heavy-used MapForce commands
- Add some more buttons and sample automation code
- Use event handlers to update command buttons

This example is available in its entirety in the file `MapForceActiveX_ApplicationLevel.htm` within the `MapForceExamples\ActiveX\HTML\` folder of your MapForce installation.

Instantiate the MapForceControl

The HTML `OBJECT` tag is used to create an instance of the MapForceControl. The Classid is that of MapForceControl. Width and height are set to 0 since we use this control as manager control without use for its user interface. The integration level is specified as a parameter within the `OBJECT` tag.

```
<OBJECT id="objMapForceX"
        Classid="clsid:A38637E9-5759-4456-A167-F01160CC22C1"
        width="0"
        height="0"
        VIEWASTEXT>
   <PARAM NAME="IntegrationLevel" VALUE="1">
</OBJECT>
```

Create Editor window

The HTML `OBJECT` tag is used to embed a document editing window. The additional custom parameter specifies that the control is to be initialized with a new empty mapping.

```
<OBJECT id="objDoc1"
        Classid="clsid:DFBB0871-DAFE-4502-BB66-08CEB7DF5255"
        width="600"
        height="500"
        VIEWASTEXT>
    <PARAM NAME="NewDocument">
</OBJECT>
```

Create Project Window

The HTML `OBJECT` tag is used to create a MapForceControlPlaceHolder window. The first additional custom parameter defines the placeholder to show the MapForce project window. The second parameter loads one of the example projects delivered coming with your MapForce installation.

```
<OBJECT id="objProjectWindow"
        Classid="clsid:FDEC3B04-05F2-427d-988C-F03A85DE53C2"
        width="200"
        height="200"
        VIEWASTEXT>
```

```
    <PARAM name="PlaceholderWindowID" value="3">
    <PARAM name="FileName" value="MapForceExamples/MapForceExamples.mfp">
</OBJECT>
```

Create Placeholder for MapForce Helper Windows

The HTML `OBJECT` tag is used to instantiate a MapForceControlPlaceHolder ActiveX control that can host the different MapForce helper window. Initially, no helper window is shown.

```
<OBJECT id="objPlaceholderWindow"
        Classid="clsid:FDEC3B04-05F2-427d-988C-F03A85DE53C2"
        width="200"
        height="200"
        VIEWASTEXT>
   <PARAM name="PlaceholderWindowID" value="0">
</OBJECT>
```

Three buttons allow us to switch the actual window that will be shown. The JavaScript execute on-button-click sets the property `PlaceHolderWindowID` to the corresponding value defined in `MapForceControlPlaceholderWindow`.

```
<input type="button" value="Library Window" onclick="BtnHelperWindow(0)">
<input type="button" value="Overview Window" onclick="BtnHelperWindow(1)">
<input type="button" value="Validation Window" onclick="BtnHelperWindow(2)">

<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
//
------------------------------------------------------------------------------
// specify which of the helper windows shall be shown in the placeholder
control.
function BtnHelperWindow(i_ePlaceholderWindowID)
{
       objPlaceholderWindow.PlaceholderWindowID = i_ePlaceholderWindowID;
}
</SCRIPT>
```

Create a Custom Toolbar

The custom toolbar consists of buttons with images of MapForce commands.

```
<button id="btnInsertXML" title="Insert XML Schema/File"
onclick="BtnDoCommand(13635)">
    <img src="..\Images\ID_INSERT_XSD.gif" width="16" height="16" />
</button>
<button id="btnInsertDB" title="Insert Database"
onclick="BtnDoCommand(13590)">
    <img src="..\Images\ID_INSERT_DATABASE.gif" width="16" height="16" />
</button>
<button id="btnInsertEDI" title="Insert EDI" onclick="BtnDoCommand(13591)">
    <img src="..\Images\ID_INSERT_EDI.gif" width="16" height="16" />
</button>
...
...
```

On clicking one of these buttons the corresponding command Id is sent to the manager control.

```
<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
// ------------------------------------------------------------------
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand(cmdID)
{
        objMapForceX.Exec( cmdID );
        msgtext.innerText = "Command " + cmdID + " performed.";
}
</SCRIPT>
```

Create More Buttons

In the example, we add some more buttons to show some automation code.

```
<p>
    <input type="button" value="New File" onclick="BtnNewFile(objDoc1)">
    <input type="button" value="Save File" onclick="BtnSaveFile(objDoc1)">
    <input type="text" title="Path" id="strPath" width="150">
    <input type="button" value="Open MarketingExpenses"
onclick="BtnOpenMEFile(objDoc1)">
</p>
<p>
    <input type="button" id="GenerateXSLT"   value="Generate XSLT"
onclick="BtnGenerate( objDoc1, 0 )">
    <input type="button" id="GenerateJava"   value="Generate Java"
onclick="BtnGenerate( objDoc1, 1 )">
    <input type="button" id="GenerateCpp"    value="Generate C++"
onclick="BtnGenerate( objDoc1, 2 )">
    <input type="button" id="GenerateCSharp" value="Generate C#"
onclick="BtnGenerate( objDoc1, 3 )">
</p>
```

The corresponding JavaScript looks like this:

```
<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
// ---------------------------------------------------------
// open a document in the specified document control window.
function BtnOpenMEFile(objDocCtrl)
{
        // do not use MapForceX.Application.OpenDocument(...) to open a
document,
        // since then MapForceControl wouldn't know a control window to show
        // the document in. Instead:

        var pos = objMapForceX.BaseHref.indexOf("ActiveX");

        if(pos > 7)
        {
                path = objMapForceX.BaseHref.substr(7, pos - 7);            //
remove file protocol

                objDocCtrl.OpenDocument(path + "MarketingExpenses.mfd");
                objDocCtrl.setActive();
        }
        else
        {
                alert("Unable to locate MarketingExpenses.mfd at: " +
objMapForceX.BaseHref);
        }
}

// -----------------------------------------------------------------
// open a new empty document in the specified document control window.
function BtnNewFile(objDocCtrl)
{
        objDocCtrl.OpenDocument("");
        objDocCtrl.setActive();
}
```

```
// ----------------------------------------------------------------------
// Saves the current file in the specified document control window.
function BtnSaveFile(objDocCtrl)
{
        if(objDocCtrl.Path.length > 0)
                objDocCtrl.SaveDocument();
        else
        {
                if(strPath.value.length > 0)
                {
                        objDocCtrl.Path = strPath.value;
                        objDocCtrl.SaveDocument();
                }
                else
                {
                        alert("Please set path for the document first!");
                        strPath.focus();
                }
        }

        objDocCtrl.setActive();
}
</SCRIPT>
```

Create Event Handler to Update Button Status

Availability of a command may vary with every mouseclick or keystroke. The custom event `OnUpdateCmdUI` of MapForceControl gives us an opportunity to update the enabled/disabled state of buttons associated with MapForce commands. The method `MapForceControl.QueryStatus` is used to query whether a command is enabled or not.

```
<SCRIPT FOR="objMapForceX" event="OnUpdateCmdUI()" LANGUAGE="javascript">
        if ( document.readyState == "complete" )            // 'complete'
        {
                // update status of buttons
                GenerateXSLT.disabled =  ! (objDoc1.QueryStatus(13617) & 0x02);
        // not enabled
                GenerateJava.disabled =  ! (objDoc1.QueryStatus(13587) & 0x02);
        // not enabled
                GenerateCpp.disabled =   ! (objDoc1.QueryStatus(13589) & 0x02);
        // not enabled
                GenerateCSharp.disabled = ! (objDoc1.QueryStatus(13588) & 0x02);
        // not enabled

                btnFuncUserDef.disabled = ! (objDoc1.QueryStatus(13633) & 0x02);
                btnFuncUserDefSel.disabled = ! (objDoc1.QueryStatus(13634) &
0x02);
                btnFuncSettings.disabled = ! (objDoc1.QueryStatus(13632) & 0x02
);
                btnInsertInput.disabled = ! (objDoc1.QueryStatus(13491) & 0x02);

                btnGenXSLT.disabled = ! (objDoc1.QueryStatus(13617) & 0x02);
                btnGenXSLT2.disabled = ! (objDoc1.QueryStatus(13618) & 0x02);
                btnGenXQuery.disabled = ! (objDoc1.QueryStatus(13586) & 0x02);
                btnGenCPP.disabled = ! (objDoc1.QueryStatus(13589) & 0x02);
                btnGenCSharp.disabled = ! (objDoc1.QueryStatus(13588) & 0x02);
                btnGenJava.disabled = ! (objDoc1.QueryStatus(13587) & 0x02);
        }

        // set activity status of simulated toolbar
</SCRIPT>
```

Visual Basic

Source code for an integration of MapForceControl into a VisualBasic program can be found in the folder `MapForceExamples\ActiveX\VisualBasic6` relative to your MapForce installation.

24.3 Command Table

Tables in this section list the names and identifiers of all commands that are available within MapForce. Every sub-section lists the commands from the corresponding top-level menu of MapForce. The left-most column shows the command's menu text to make it easier for you to identify the functionality behind the command. The last sub-section is a collection of those commands that are not accessible via the main menu.

Depending on the edition of MapForce you have installed, some of these commands might not be supported. See `Query MapForce Commands` on how to query the current resource structure and command availability. The same topics shows how to use the same command icons that are used by MapForce if you are not already integrating on application-level.

Use the command identifiers with `MapForceControl.QueryStatus` or `MapForceControlDocument.QueryStatus` to check the current status of a command. Use `MapForceControl.Exec` or `MapForceControlDocument.Exec` to execute a command.

24.3.1 File Menu

Commands from the File menu:

Menu Text	Command Name	ID
New...	ID_FILE_NEW	32373
Open...	ID_FILE_OPEN	32374
Save	ID_FILE_SAVE	32376
Save As...	ID_FILE_SAVE_AS	32379
Save All	ID_FILE_SAVEALL	32377
Close	ID_WINDOW_CLOSE	32453
Close All	ID_WINDOW_CLOSEALL	32454
Save Project	ID_FILE_SAVEPROJECT	32378
Close Project	ID_FILE_CLOSEPROJECT	32355
Print...	IDC_FILE_PRINT	32319
Print Preview	IDC_FILE_PRINT_PREVIEW	32320
Print Setup...	ID_FILE_PRINT_SETUP	32375
Generate code in selected language	ID_FILE_GENERATE_SELECTED_CODE	32362
Generate code in/XSLT 1.0	ID_FILE_GENERATEXSLT	32360
Generate code in/XSLT 2.0	ID_FILE_GENERATEXSLT2	32361
Generate code in/XQuery	ID_FILE_GENERATEXQUERY	32359
Generate code in/Java	ID_FILE_GENERATEJAVACODE	32358
Generate code in/C# (Sharp)	ID_FILE_GENERATECSCODE	32357
Generate code in/C++	ID_FILE_GENERATECPPCODE	32356
Mapping Settings...	ID_MAPPING_SETTINGS	32396
Recent Files/Recent File	ID_FILE_MRU_FILE1	32363
Recent Projects/Recent Project	ID_FILE_MRU_PROJECT1	32364
Exit	ID_APP_EXIT	32333

24.3.2 Edit Menu

Commands from the Edit menu:

Menu Text	Command Name	ID
Undo	ID_EDIT_UNDO	32354
Redo	ID_EDIT_REDO	32352
Find...	ID_EDIT_FIND	32348
Find next	ID_EDIT_FINDNEXT	32349
Cut	ID_EDIT_CUT	32346
Copy	ID_EDIT_COPY	32345
Paste	ID_EDIT_PASTE	32351
Delete	ID_EDIT_CLEAR	32347
Select All	ID_EDIT_SELECT_ALL	32353

24.3.3 Insert Menu

Commands from the Insert menu:

Menu Text	Command Name	ID
XML Schema/File	ID_INSERT_XSD	32393
Database	ID_INSERT_DATABASE	32389
EDI	ID_INSERT_EDI	32390
Text file	ID_INSERT_TXT	32392
Constant	ID_INSERT_CONSTANT	32388
Filter: Nodes/Rows	ID_INSERT_FILTER	32391
IF-Else Condition	ID_INSRT_CONDITION	32394
Exception	ID_INSERT_EXCEPTION	32311

24.3.4 Project Menu

Commands from the Project menu:

Menu Text	Command Name	ID
Add Files to Project...	ID_PROJECT_ADDFILESTOPROJECT	32420
Add Active File to Project...	ID_PROJECT_ADDACTIVEFILETOPROJECT	32419
Create Folder	ID_POPUP_PROJECT_CREATE_FOLDER	32310
Open Mapping for Operation	ID_POPUP_OPENOPERATIONSMAPPING	13692
Create Mapping for Operation...	ID_POPUP_CREATEMAPPINGFOROPERATION	32399
Add Mapping File for Operation...	ID_POPUP_PROJECT_ADD_MAPPING	32309
Remove Item	ID_PROJECT_REMOVE_ITEM	32415
Insert Web Service...	ID_POPUP_PROJECT_INSERT_WEBSERVICE	32306
Open WSDL file In XMLSpy	ID_POPUP_PROJECT_OPENINXMLSPY	32305
Generate Code for Entire Project	ID_POPUP_PROJECT_GENERATE_PROJECT	32304
Generate code in/XSLT 1.0	ID_PROJECT_GENERATEXSLT	32425
Generate code in/XSLT 2.0	ID_PROJECT_GENERATEXSLT2	32426
Generate code in/XQuery	ID_PROJECT_GENERATEXQUERY	32424
Generate code in/Java	ID_PROJECT_GENERATEJAVACODE	32423
Generate code in/C# (Sharp)	ID_PROJECT_GENERATECSCODE	32422
Generate code in/C++	ID_PROJECT_GENERATECPPCODE	32421
Project Settings...	ID_PROJECT_PROPERTIES	32404

24.3.5 Component Menu

Commands from the Component menu:

Menu Text	Command Name	ID
Edit Constant	ID_COMPONENT_EDIT_CONSTANT	32336
Align Tree Left	ID_COMPONENT_LEFTALIGNTREE	32338
Align Tree Right	ID_COMPONENT_RIGHTALIGNTREE	32340
Change Root Element	ID_COMPONENT_CHANGEROOTELEMENT	32334
Edit Schema Definition in XMLSpy	ID_COMPONENT_EDIT_SCHEMA	32337
Duplicate Input	ID_COMPONENT_CREATE_DUPLICATE_ICON	32335
Remove Duplicate	ID_COMPONENT_REMOVE_DUPLICATE_ICON	32339
Database Table Actions	ID_POPUP_DATABASETABLEACTIONS	32400
Database Key Settings	ID_POPUP_VALUEKEYSETTINGS	32417
Component Settings...	ID_COMPONENT_SETTINGS	32341

24.3.6 Connection Menu

Commands from the Connection menu:

Menu Text	Menu Text	ID
Auto Connect Matching Children	ID_CONNECTION_AUTOCONNECTCHILDREN	32342
Settings for Connect Matching Children...	ID_CONNECTION_SETTINGS	32344
Connect Matching Children	ID_CONNECTION_MAPCHILDELEMENTS	32343
Standard Mapping (target driven)	ID_POPUP_NORMALCONNECTION	32401
Source-driven Mapping (mixed-content)	ID_POPUP_ORDERBYSOURCECONNECTION	32403
Connection Settings...	ID_POPUP_CONNECTION_SETTINGS	32398

24.3.7 Function Menu

Commands from the Function menu:

Menu Text	Command Name	ID
Create User-Defined Function...	ID_FUNCTION_CREATE_EMPTY	32380
Create User-Defined Function From Selection...	ID_FUNCTION_CREATE_FROM_SELECTION	32381
Function Settings...	ID_FUNCTION_SETTINGS	32387
Insert Input	ID_FUNCTION_INSERT_INPUT	32383
Insert Output...	ID_FUNCTION_INSERT_OUTPUT	32402

24.3.8 Output Menu

Commands from the Output menu:

Menu Text	Command Name	ID
XSLT 1.0	ID_SELECT_LANGUAGE_XSLT	32433
XSLT 2.0	ID_SELECT_LANGUAGE_XSLT2	32434
XQuery	ID_SELECT_LANGUAGE_XQUERY	32432
Java	ID_SELECT_LANGUAGE_JAVA	32431
C# (Sharp)	ID_SELECT_LANGUAGE_CSHARP	32430
C++	ID_SELECT_LANGUAGE_CPP	32429
Validate output XML file	ID_XML_VALIDATE	32458
Save Output File...	IDC_FILE_SAVEGENERATEDOUTPUT	32321
Run SQL-script	ID_TRANSFORM_RUN_SQL	32442

24.3.9 View Menu

Commands from the View menu:

Menu Text	Command Name	ID
Show Annotations	ID_SHOW_ANNOTATION	32435
Show Types	ID_SHOW_TYPES	32437
Show Library In Function Header	ID_VIEW_SHOWLIBRARYINFUNCTIONHEADER	32448
Show Tips	ID_SHOW_TIPS	32436
Show selected component connectors	ID_VIEW_AUTOHIGHLIGHTCOMPONENTCONNECTIONS	32443
Show connectors from source to target	ID_VIEW_RECURSIVEAUTOHIGHLIGHT	32447
Zoom	ID_VIEW_ZOOM	32451
Status Bar	ID_VIEW_STATUS_BAR	32449
Library Window	ID_VIEW_LIBRARY_WINDOW	32445
Validation Output	ID_VIEW_VALIDATION_OUTPUT	32450
Overview	ID_VIEW_OVERVIEW_WINDOW	32446
Project Window	ID_VIEW_PROJECT_WINDOW	32302

24.3.10 Tools Menu

Commands from the Tools menu:

Menu Text	Command Name	ID
Customize...	ID_VIEW_CUSTOMIZE	32444
Options...	ID_TOOLS_OPTIONS	32441

24.3.11 Window Menu

Commands from the Window menu:

Menu Text	Command Name	ID
Close	ID_WINDOW_CLOSE	32453
Close All	ID_WINDOW_CLOSEALL	32454
Cascade	ID_WINDOW_CASCADE	32452
Tile Horizontal	ID_WINDOW_TILE_HORZ	32455
Tile Vertical	ID_WINDOW_TILE_VERT	32456
Close	ID_WINDOW_CLOSE	32453
Close All	ID_WINDOW_CLOSEALL	32454

24.3.12 Help Menu

Commands from the Help menu:

Menu Text	Command Name	ID
Table of Contents...	IDC_HELP_CONTENTS	32322
Index..	IDC_HELP_INDEX	32323
Search...	IDC_HELP_SEARCH	32324
Registration...	IDC_REGISTRATION	32330
Order Form...	IDC_OPEN_ORDER_PAGE	32326
Support Center...	IDC_OPEN_SUPPORT_PAGE	32327
FAQ on the Web...	IDC_SHOW_FAQ	32331
Components Download...	IDC_OPEN_COMPONENTS_PAGE	32325
MapForce on the Internet..	IDC_OPEN_XML_SPY_HOME	32328
MapForce Training...	IDC_OPEN_MAPFORCE_TRAINING_PAGE	32300
About MapForce...	ID_APP_ABOUT	32332

24.3.13 Commands not in Main Menu

Commands not in the main menu:

Menu Text	Command Name	ID
	IDC_QUICK_HELP	32329
Edit FlexText Configuration	ID_COMPONENT_EDIT_MFT	32301
Priority Context	ID_COMPONENT_PRIORITYCONTEXT	32318
	ID_EDIT_FINDPREV	32350
	ID_FUNCTION_GOTO_MAIN	32382
Insert Input	ID_FUNCTION_INSERT_INPUT_AT_POINT	32384
	ID_FUNCTION_REMOVE	32385
Replace component with internal function structure	ID_FUNCTION_REPLACE_WITH_COMPONENTS	32386
	ID_MAPFORCEVIEW_ZOOM	32395
	ID_NEXT_PANE	32397
Add Active File to Project	ID_POPUP_PROJECT_ADDACTIVEFILETOPROJECT	32405
Add Files to Project...	ID_POPUP_PROJECT_ADDFILESTOPROJECT	32406
C++	ID_POPUP_PROJECT_GENERATECPPCODE	32408
C# (Sharp)	ID_POPUP_PROJECT_GENERATECSCODE	32409
Java	ID_POPUP_PROJECT_GENERATEJAVACODE	32410
XQuery	ID_POPUP_PROJECT_GENERATEXQUERY	32411
XSLT 1.0	ID_POPUP_PROJECT_GENERATEXSLT	32412
XSLT 2.0	ID_POPUP_PROJECT_GENERATEXSLT2	32413
Generate All	ID_POPUP_PROJECT_GENERATE_ALL	32303
Generate code in default language	ID_POPUP_PROJECT_GENERATE_CODE	32414
Open	ID_POPUP_PROJECT_OPEN_MAPPING	32307
Properties...	ID_POPUP_PROJECT_PROJECTPROPERTIES	32428
Remove	ID_POPUP_PROJECT_REMOVE	32308
	ID_PREV_PANE	32418
	ID_TOGGLE_FOLDINGMARGIN	32438
	ID_TOGGLE_INDENTGUIDES	32439
	ID_TOGGLE_NUMLINEMARGIN	32440
	ID_WORD_WRAP	32457

24.4 Accessing MapForce API

The focus of this documentation is the ActiveX controls and interfaces required to integrate the MapForce user interface into your application. To allow you to automate or control the functionality of the integrated components, the following properties give you access to the MapForce automation interface (MapForce API):

```
MapForceControl.Application
MapForceControlDocument.Document
MapForceControlPlaceHolder.Project
```

Some restrictions apply to the usage of the MapForce automation interface when integrating MapForceControl at document-level. See Integration at document level for details.

24.5 Object Reference

Objects:
MapForceCommand
MapForceCommands
MapForceControl
MapForceControlDocument
MapForceControlPlaceHolder

To give access to standard MapForce functionality, objects of the **MapForce automation interface** can be accessed as well. See MapForceControl.Application, MapForceControlDocument.Document and MapForceControlPlaceHolder.Project for more information.

24.5.1 MapForceCommand

Properties:
`ID`
`Label`
`IsSeparator`
`SubCommands`

Description:
Each `MapForceCommand` object can be one of three possible types:

- **Command**: `ID` is set to a value greater `0` and `Label` is set to the command name. `IsSeparator` is `false` and the `SubCommands` collection is empty.
- **Separator**: `IsSeparator` is `true`. `ID` is `0` and `Label` is not set. The `SubCommands` collection is empty.
- (**Sub**) **Menu**: The `SubCommands` collection contains `MapForceCommand` objects and `Label` is the name of the menu. `ID` is set to `0` and `IsSeparator` is false.

ID

Property: `ID` as long

Description:
`ID` is `0` for separators and menus.
For commands, this is the ID which can be used with `Exec` and `QueryStatus`.

Label

Property: `Label` as string

Description:
Label is empty for separators.
For command objects that are children of the ALL_COMMANDS collection, this is a unique name. Command icons are stored in files with this name. See Query MapForceCommands for more information.
For command objects that are children of menus, the label property holds the command's menu text.
For sub-menus, this property holds the menu text.

IsSeparator

Property: `IsSeparator` as boolean

Description:
True if the command is a separator.

SubCommands

Property: `SubCommands` as `MapForceCommands`

Description:
The `SubCommands` collection holds any sub-commands if this command is actually a menu or submenu.

24.5.2 MapForceCommands

Properties:
`Count`
`Item`

Description:
Collection of `MapForceCommand` objects to get access to command labels and IDs of the MapForceControl. Those commands can be executed with the `Exec` method and their status can be queried with `QueryStatus`.

Count

Property: `Count` as long

Description:
Number of `MapForceCommand` objects on this level of the collection.

Item

Property: `Item` (n as long) as `MapForceCommand`

Description:
Gets the command with the index `n` in this collection. Index is 1-based.

24.5.3 MapForceControl

Properties:
`IntegrationLevel`
`Appearance`
`Application`
`BorderStyle`
`CommandsList`
`CommandsStructure (deprecated)`
`EnableUserPrompts`
`MainMenu`
`ReadOnly`
`Toolbars`

Methods:
`Open`
`Exec`
`QueryStatus`

Events:
`OnDocumentOpened`
`OnProjectOpened`
`OnUpdateCmdUI`
`OnCloseEditingWindow`

This object is a complete ActiveX control and should only be visible if the MapForce library is used in the Application Level mode.

```
CLSID: A38637E9-5759-4456-A167-F01160CC22C1
ProgID: Altova.MapForceControl
```

Properties

The following properties are defined:

`IntegrationLevel`
`ReadOnly`
`EnableUserPrompts`
`Appearance`
`BorderStyle`

Command related properties:
`CommandsList`
`MainMenu`
`Toolbars`
`CommandsStructure (deprecated)`

Access to MapForceAPI:
`Application`

Appearance

Property: `Appearance` as short

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the control. Default value is `0`.

Application

Property: `Application` as Application

Dispatch Id: *4*

Description:
The `Application` property gives access to the `Application` object of the complete MapForce automation server API. The property is read-only.

BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

CommandsList

Property: `CommandList` as `MapForceCommands` (read-only)

Dispatch Id: *1004*

Description:
This property returns a flat list of all commands defined available with MapForceControl. For more information see C# Sample.

CommandsStructure (deprecated)

Property: `CommandsStructure` as `MapForceCommands` (deprecated)

Dispatch Id: *3*

Remark:
This property is deprecated. Instead, use `CommandsList`, `MainMenu`, `Toolbars`.

Description:
The `CommandsStructure` collection contains all commands of the MapForceControl as `MapForceCommand` objects. At the first level of the collection two special `MapForceCommand` objects with the following labels are accessible:

- `IDR_MAPFORCE`: This object holds all commands as hierarchical menu structure.
- `ALL_COMMANDS`: This object holds all commands in a flat list.

Sample:
C# code to access the first level of the collection.

```
MapForceCommands  objCommands;
objCommands = axMapForceControl.CommandsStructure;

long nCount = objCommands.Count;

for(long idx = 0;idx < nCount;idx++)
```

```
{
    MapForceCommand objCommand;
    objCommand = objCommands[(int)idx];

    // We are looking for the Menu with the name IDR_MAPFORCE. This
    menu should contain
    // the complete main menu of MapForce.

    if(objCommand.Label == "IDR_MAPFORCE")
    {
        // read menu structure here...
    }

    if(objCommand.Label == "ALL_COMMANDS")
    {
        // read all commands here...
    }
}
```

EnableUserPrompts

Property: `EnableUserPrompts` as boolean

Dispatch Id: *1006*

Description:
Setting this property to *false*, disables user prompts in the control. The default value is *true*.

IntegrationLevel

Property: `IntegrationLevel` as `ICActiveXIntegrationLevel`

Dispatch Id: *1000*

Description:
The `IntegrationLevel` property determines the operation mode of the control. See also Integration at the application level and Integration at document level for more information.

Note: It is important to set this property immediately after the creation of the `MapForceControl` object.

MainMenu

Property: `MainMenu as MapForceCommand` (read-only)

Dispatch Id: *1003*

Description:
This property gives access to the description of the MapForceControl main menu. For more information see C# Sample.

ReadOnly

Property: `ReadOnly` as boolean

Dispatch Id: *2*

Description:
Using this property you can turn on and off the read-only mode of the control. If ReadOnly is true it is not possible to modify any document loaded. This property is only used in the Application-level integration mode.

Toolbars

Property: `Toolbars` as `MapForceCommands` (read-only)

Dispatch Id: *1005*

Description:
This property returns a list of all toolbar descriptions that describe all toolbars available with MapForceControl. For more information see C# Sample.

Methods

The following methods are defined:

`Open`
`Exec`
`QueryStatus`

Exec

Method: `Exec` (`nCmdID` as long) as boolean

Dispatch Id: *6*

Description:
`Exec` calls the `MapForce` command with the ID `nCmdID`. If the command can be executed, the method returns `true`. See also `CommandsStructure` to get a list of all available commands and `QueryStatus` to retrieve the status of any command.

Open

Method: `Open` (`strFilePath` as string) as boolean

Dispatch Id: *5*

Description:
The result of the method depends on the extension passed in the argument `strFilePath`. If the file extension is `.mfd`, a new document is opened. If the file extension is `.mfp`, the corresponding project is opened. If a different file extension is passed into the method, the control tries to load the file as a new component into the active document.

Do not use this method to load documents or projects when using the control in document-level integration mode. Instead, use `MapForceControlDocument.OpenDocument` and `MapForceControlPlaceHolder.OpenProject`.

QueryStatus

Method: `QueryStatus` (`nCmdID` as long) as long

Dispatch Id: *7*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5`, the command is disabled.

Events

The MapForceControl ActiveX control provides the following connection point events:

`OnUpdateCmdUI`
`OnDocumentOpened`
`OnProjectOpened`
`OnCloseEditingWindow`

OnDocumentOpened

Event: `OnDocumentOpened` (`objDocument` as Document)

Dispatch Id: *3*

Description:
This event gets triggered whenever a document gets opened. The argument `objDocument` is a `Document` object from the MapForce automation interface and can be used to query more details on the document or perform additional operations. When integrating on document-level it is often better to use the event `MapForceControlDocument.OnDocumentOpened` instead.

OnCloseEditingWindow

Event: `OnCloseEditingWindow` (i_strFilePath as String) as boolean

Dispatch Id: *1002*

Description:
This event gets triggered when MapForce needs to close an already open document. As an answer to this event, clients should close the editor window associated with *i_strFilePath*. Returning *true* from this event indicates that the client has closed the document. Clients can return *false* if no specific handling is required and MapForceControl should try to close the editor and destroy the associated document control.

OnProjectOpened

Event: `OnProjectOpened` (`objProject` as Project)

Dispatch Id: *2*

Description:
This event gets triggered whenever a project is opened. The argument `objProjectDocument` is a `Project` object from the MapForce automation interface and can be used to query more details on the project or perform additional operations.

OnUpdateCmdUI

Event: `OnUpdateCmdUI ()`

Dispatch Id: *1*

Description:
Called frequently to give integrators a good opportunity to check status of MapForce commands using `MapForceControl.QueryStatus`. Do not perform long operations in this callback.

24.5.4 MapForceControlDocument

Properties:
`Appearance`
`BorderStyle`
`Document`
`IsModified`
`Path`
`ReadOnly`
`ZoomLevel`

Methods:
`Exec`
`New`
`Open`
`QueryStatus`
`Reload`
`Save`
`SaveAs`

If the MapForceControl is integrated in the Document Level mode each document is displayed in an own object of type `MapForceControlDocument`. The `MapForceControlDocument` contains only one document at the time but can be reused to display different files one after another.

This object is a complete ActiveX control.

```
CLSID:  DFBB0871-DAFE-4502-BB66-08CEB7DF5255
ProgID:  Altova.MapForceControlDocument
```

Properties

The following properties are defined:

`ReadOnly`
`IsModified`
`ZoomLevel`
`Path`
`Appearance`
`BorderStyle`

Access to MapForceAPI:
`Document`

Appearance

Property: `Appearance` as short

Dispatch Id: *-520*

Description:
A value not equal to `0` displays a client edge around the document control. Default value is `0`.

BorderStyle

Property: `BorderStyle` as short

Dispatch Id: *-504*

Description:
A value of `1` displays the control with a thin border. Default value is `0`.

Document

Property: `Document` as Document

Dispatch Id: *3*

Description:
The `Document` property gives access to the `Document` object of the MapForce automation server API. This interface provides additional functionalities which can be used with the document loaded in the control. The property is read-only.

IsModified

Property: `IsModified` as boolean (read-only)

Dispatch Id: *1006*

Description:
`IsModified` is *true* if the document content has changed since the last open, reload or save operation. It is *false*, otherwise.

Path

Property: `Path` as string

Dispatch Id: *1005*

Description:
Sets or gets the full path name of the document loaded into the control.

ReadOnly

Property: `ReadOnly` as boolean

Dispatch Id: *1007*

Description:
Using this property you can turn on and off the read-only mode of the document. If `ReadOnly` is `true` it is not possible to do any modifications.

ZoomLevel

Property: `ZoomLevel` as long

Dispatch Id: *1002*

Description:
The `ZoomLevel` property allows to set the Mapping view magnification in a range from 1 to 100. A `ZoomLevel` of 50 is the default and shows the view content at normal size.

Methods

The following methods are defined:

Document handling:
New
Open
Reload
Save
SaveAs
OpenDocument (deprecated)
NewDocument (deprecated)
SaveDocument (deprecated)

Command Handling:
`Exec`
`QueryStatus`

Exec

Method: `Exec (nCmdID` as long) as boolean

Dispatch Id: *8*

Description:
`Exec` calls the MapForce command with the ID `nCmdID`. If the command can be executed, the method returns `true`. The client should call the `Exec` method of the document control if there is currently an active document available in the application.

See also `CommandsStructure` to get a list of all available commands and `QueryStatus` to retrieve the status of any command.

New

Method: `New` () as boolean

Dispatch Id: *1000*

Description:
This method initializes a new mapping inside the control..

NewDocument (deprecated)

Method: `NewDocument ()` as boolean (deprecated)

Description:
The method resets the content of the `MapForceControlDocument` object to a new empty document. Please use the `Path` property to set path and filename. Otherwise the control can't save the document using `SaveDocument`.

Open

Method: `Open` (`strFileName` as string) as boolean

Dispatch Id: *1001*

Description:
`Open` loads the file `strFileName` as the new document into the control.

OpenDocument (deprecated)

Method: `OpenDocument` (`strFileName` as string) as boolean (deprecated)

Description:
`OpenDocument` loads the file `strFileName` as the new document into the control.

QueryStatus

Method: `QueryStatus` (`nCmdID` as long) as long

Dispatch Id: *9*

Description:
`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns `0` the command ID is not recognized as a valid MapForce command. If `QueryStatus` returns a value of `1` or `5` the command is disabled. The client should call the `QueryStatus` method of the document control if there is currently an active document available in the application.

Reload

Method: `Reload` () as boolean

Dispatch Id: *1002*

Description:
`Reload` updates the document content from the file system.

Save

Method: `Save` () as boolean

Dispatch Id: *1003*

Description:
`Save` saves the current document at the location `Path`.

SaveAs

Method: `OpenDocument` (`strFileName` as string) as boolean

Dispatch Id: *1004*

Description:
`SaveAs` sets [`Path`] to *strFileName* and then saves the document to this location.

SaveDocument (deprecated)

Method: `SaveDocument ()` as boolean (deprecated)

Description:
`SaveDocument` saves the current document at the location `Path`.

Events

The MapForceControlDocument ActiveX control provides following connection point events:

`OnDocumentOpened`
`OnDocumentClosed`
`OnModifiedFlagChanged`

OnDocumentClosed

Event: `OnDocumentClosed` (`objDocument` as Document)

Dispatch Id: *2*

Description:
This event gets triggered whenever the document loaded into this control gets closed. The argument `objDocument` is a `Document` object from the MapForce automation interface and should be used with care.

OnDocumentOpened

Event: `OnDocumentOpened` (`objDocument` as Document)

Dispatch Id: *1*

Description:
This event gets triggered whenever a document gets opened in this control. The argument `objDocument` is a `Document` object from the MapForce automation interface and can be used to query more details about the document or perform additional operations.

OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as boolean)

Dispatch Id: *3*

Description:
This event gets triggered whenever the document changes between modified and unmodified state. The parameter *i_bIsModifed* is *true* if the document contents differs from the original content, and *false*, otherwise.

24.5.5 MapForceControlPlaceHolder

Properties available for all kinds of placeholder windows:
PlaceholderWindowID

Properties for project placeholder window:
Project

Methods for project placeholder window:
OpenProject

The `MapForceControlPlaceHolder` control is used to show the additional MapForce windows like Overview, Library or Project window. It is used like any other ActiveX control and can be placed anywhere in the client application.

```
CLSID: FDEC3B04-05F2-427d-988C-F03A85DE53C2
ProgID: Altova.MapForceControlPlaceHolder
```

Properties

The following properties are defined:

PlaceholderWindowID

Access to MapForceAPI:
Project

PlaceholderWindowID

Property: `PlaceholderWindowID` as MapForceControlPlaceholderWindow

Dispatch Id: *1*

Description:
Using this property the object knows which MapForce window should be displayed in the client area of the control. The `PlaceholderWindowID` can be set at any time to any valid value of the MapForceControlPlaceholderWindow enumeration. The control changes its state immediately and shows the new MapForce window.

Project

Property: `Project` as Project (read-only)

Dispatch Id: *2*

Description:
The `Project` property gives access to the `Project` object of the MapForce automation server API. This interface provides additional functionalities which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has PlaceholderWindowID with a value of `MapForceXProjectWindow (=3)`. The property is read-only.

Methods

The following method is defined:

OpenProject

OpenProject

Method: `OpenProject` (`strFileName` as string) as boolean

Dispatch Id: *3*

Description:
`OpenProject` loads the file `strFileName` as the new project into the control. The method will fail if the placeholder window has a `PlaceholderWindowID` different to `MapForceXProjectWindow (=3)`.

Events

The MapForceControlPlaceholder ActiveX control provides following connection point events:

`OnModifiedFlagChanged`

OnModifiedFlagChanged

Event: `OnModifiedFlagChanged` (`i_bIsModified` as boolean)

Dispatch Id: *1*

Description:
This event gets triggered only for placeholder controls with a `PlaceholderWindowID` of `MapForceXProjectWindow (=3).` Th event is fired whenever the project content changes between modified and unmodified state. The parameter *i_bIsModifed* is *true* if the project contents differs from the original content, and *false*, otherwise.

24.5.6 Enumerations

The following enumerations are defined:

```
ICActiveXIntegrationLevel
MapForceControlPlaceholderWindow
```

ICActiveXIntegrationLevel

Possible values for the IntegrationLevel property of the MapForceControl.

```
ICActiveXIntegrationOnApplicationLevel    = 0
ICActiveXIntegrationOnDocumentLevel       = 1
```

MapForceControlPlaceholderWindow

This enumeration contains the list of the supported additional MapForce windows.

```
MapForceXNoWindow               = -1
MapForceXLibraryWindow          = 0
MapForceXOverviewWindow         = 1
MapForceXValidationWindow       = 2
MapForceXProjectWindow          = 3
```

Chapter 25

Appendices

25 Appendices

These appendices contain technical information about MapForce and important licensing information. Each appendix contains sub-sections as given below:

Technical Data

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage
- License metering

License Information

- Electronic software distribution
- Copyrights
- End User License Agreement

25.1 Engine information

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.

25.1.1 XSLT 1.0 Engine: Implementation Information

The Altova XSLT 1.0 Engine is built into Altova's XMLSpy, StyleVision, Authentic, and MapForce XML products. It is also available in the free AltovaXML package. The Altova XSLT 1.0 Engine implements and conforms to the World Wide Web Consortium's XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999 . Limitations and implementation-specific behavior are listed below.

Limitations

- The **`xsl:preserve-space`** and **`xsl:strip-space`** elements are not supported.
- When the `method` attribute of `xsl:output` is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document directly as special characters; they are not inserted as HTML character references in the output. For instance, the character `&#160;` (the decimal character reference for a non-breaking space) is not inserted as `&nbsp;` in the HTML code, but directly as a non-breaking space.

Implementation's handling of whitespace-only nodes in source XML document

The XML data (and, consequently, the XML Infoset) that is passed to the Altova XSLT 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a child whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping may have an effect on the value returned by the `fn:position()`, `fn:last()`, and `fn:count()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, and `fn:count()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3]`, which is short for `para[position()=3]`), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</>.</para> or
<para>This is <b>bold&#x20;</b> <i>italic</>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

25.1.2 XSLT 2.0 Engine: Implementation Information

The Altova XSLT 2.0 Engine is built into Altova's XMLSpy, StyleVision, Authentic, and MapForce XML products. It is also available in the free AltovaXML package. This section describes the engine's implementation-specific aspects of behavior. It starts with a section giving general information about the engine, and then goes on to list the implementation-specific behavior of XSLT 2.0 functions.

For information about implementation-specific behavior of XPath 2.0 functions, see the section, XPath 2.0 and XQuery 1.0 Functions.

General Information

The Altova XSLT 2.0 Engine conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Candidate Recommendation of 8 June 2006. Note the following general information about the engine.

Backwards Compatibility

The Altova XSLT 2.0 Engine is not backwards compatible. Depending on the Altova product you are using, the following options are available:

- If you wish to run an XSLT 1.0 transformation using AltovaXML, then you should use the Altova XSLT 1.0 Engine of this package.
- If you are running an XSLT transformation, or carrying out an action involving an XSLT transformation, within the XMLSpy, StyleVision, Authentic, or MapForce products, the correct built-in Altova XSLT Engine (1.0 or 2.0) is automatically selected by the application. This selection is based on the value of the `version` attribute of the `stylesheet` or `transform` element of the stylesheet.

Namespaces

Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   ...
</xsl:stylesheet>
```

The following points should be noted:

- The Altova XSLT 2.0 Engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can

additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- With the CRs of 8 June 2006, the `untypedAtomic` and duration datatypes (`dayTimeDuration` and `yearMonthDuration`), which were formerly in the XPath Datatypes namespace (typically prefixed `xdt:`) have been moved to the XML Schema namespace.
- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness

The Altova XSLT 2.0 Engine is schema-aware.

Whitespace in XML document

By default, the Altova XSLT 2.0 Engine strips all boundary whitespace from boundary-whitespace-only nodes in the source XML document. The removal of this whitespace affects the values that the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions return. For more details, see Whitespace-only Nodes in XML Document in the XPath 2.0 and XQuery 1.0 Functions section.

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</>.</para> or
<para>This is <b>bold&#x20;</b> <i>italic</>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

XSLT 2.0 elements and functions

Limitations and implementation-specific behavior of XSLT 2.0 elements and functions are listed in the section XSLT 2.0 Elements and Functions.

XPath 2.0 functions

Implementation-specific behavior of XPath 2.0 functions is listed in the section XPath 2.0 and XQuery 1.0 Functions.

XSLT 2.0 Elements and Functions

Limitations

The **`xsl:preserve-space`** and **`xsl:strip-space`** elements are not supported.

Implementation-specific behavior

Given below is a description of how the Altova XSLT 2.0 Engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

`format-date, format-dateTime, formatTime`

Presentation modifiers and formatting tokens in the variable markers of the Picture argument are not supported and, if supplied, are ignored. The optional Language, Calendar, and Country arguments are not supported and, if supplied, are ignored. Days and weeks are returned as numbers; in the case of single digit numbers, there is no preceding zero. The component specifier `F` is returned as a number. Weeks of the month are reckoned from Monday to Friday. The component specifier `P` returns `am` or `pm` (in English).

`function-available`

The function tests for the availability of XSLT 2.0 functions, not for the availability of XPath 2.0 functions.

`unparsed-text`

The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol.

25.1.3 XQuery 1.0 Engine: Implementation Information

The Altova XQuery 1.0 Engine is built into Altova's XMLSpy and MapForce XML products. It is also available in the free AltovaXML package. This section provides information about implementation-defined aspects of behavior.

Standards conformance
The Altova XQuery 1.0 Engine conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Candidate Recommendation of 8 June 2006. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the Altova XQuery 1.0 Engine implements these features.

Schema awareness
The Altova XQuery 1.0 Engine is **schema-aware**.

Encoding
The UTF-8 and UTF-16 character encodings are supported.

Namespaces
The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
Schema instance	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
Built-in functions	`fn:`	`http://www.w3.org/2005/xpath-functions`
Local functions	`local:`	`http://www.w3.org/2005/xquery-local-functions`

The following points should be noted:

- The Altova XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 8 June 2006, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation

XML documents used in executing an XQuery document with the Altova XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. No type checking is done in the static analysis phase. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules

Library modules store functions and variables so they can be reused. The Altova XQuery 1.0 Engine supports modules that are stored in **a single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns: webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at
   "modulefilename.xq";
if    ($modlib:company = "Altova")
then   modlib:webaddress()
else   error("No match found.")
```

External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations

The default collation is the Unicode codepoint collation. No other collation is currently supported. Comparisons, including the `fn:max` function, are based on this collation.

Character normalization

No character normalization form is supported.

Precision of numeric types

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support

The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

XQuery Functions Support

For information about implementation-specific behavior of XQuery 1.0 functions, see the section, XPath 2.0 and XQuery 1.0 Functions.

25.1.4 XPath 2.0 and XQuery 1.0 Functions

XPath 2.0 and XQuery 1.0 functions are evaluated by:

- the **Altova XPath 2.0 Engine**, which (i) is a component of the Altova XSLT 2.0 Engine, and (ii) is used in the XPath Evaluator of Altova's XMLSpy product to evaluate XPath expressions with respect to the XML document that is active in the XMLSpy interface.
- the **Altova XQuery 1.0 Engine**.

This section describes how XPath 2.0 and XQuery 1.0 functions are handled by the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine. Only those functions are listed, for which the behavior is implementation-specific, or where the behavior of an individual function is different in any of the three environments in which these functions are used (that is, in XSLT 2.0, in XQuery 1.0, and in the XPath Evaluator of XMLSpy). Note that this section does not describe how to use these functions. For more information about the usage of functions, see the World Wide Web Consortium's (W3C's) XQuery 1.0 and XPath 2.0 Functions and Operators CR of 8 June 2006.

General Information

Standards conformance

- The Altova XPath 2.0 Engine implements the World Wide Web Consortium's (W3C's) XPath 2.0 Candidate Recommendation of 8 June 2006. The Altova XQuery 1.0 Engine implements the World Wide Web Consortium's (W3C's) XQuery 1.0 Candidate Recommendation of 8 June 2006. The XPath 2.0 and XQuery 1.0 functions support in these two engines is compliant with the XQuery 1.0 and XPath 2.0 Functions and Operators CR of 8 June 2006.
- The Altova XPath 2.0 Engine conforms to the rules of XML 1.0 (Third Edition) and XML Namespaces (1.0).

Default functions namespace

The default functions namespace has been set to comply with that specified in the standard. Functions can therefore be called without a prefix.

Boundary-whitespace-only nodes in source XML document

The XML data (and, consequently, the XML Infoset) that is passed to the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a child whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping has an effect on the value returned by the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3],` which is short for `para[position()`=3]), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

Numeric notation

On output, when an `xs:double` is converted to a string, scientific notation (for example, `1.0E12`) is used when the absolute value is less than 0.000001 or greater than 1,000,000. Otherwise decimal or integer notation is used.

Precision of `xs:decimal`

The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezone of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `fn:implicit-timezone()` function.

Collations

Only the Unicode codepoint collation is supported. No other collations can be used. String comparisons, including for the `fn:max` and `fn:min` functions, are based on this collation.

Namespace axis

The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `fn:in-scope-prefixes()`, `fn:namespace-uri()` and `fn:namespace-uri-for-prefix()` functions.

Static typing extensions

The optional static type checking feature is not supported.

Functions Support

The table below lists (in alphabetical order) the implementation-specific behavior of certain functions. The following general points should be noted:

- In general, when a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

Function Name	Notes

`base-uri`	• If external entities are used in the source XML document and if a node in the external entity is specified as the input node argument of the `base-uri()` function, it is still the base URI of the including XML document that is used—not the base URI of the external entity. • The use of the `xml:base` attribute in the source XML document is not supported. This means that the base URI of a node in the XML document cannot be altered using the `xml:base` attribute.
`collection`	• The `collection()` function is a mapping of form `(string, nodes)`, currently called `available collections` and left empty by the external environment. The function therefore returns either (i) the empty sequence (when called with no argument or with an empty sequence), or (ii) an error (when called with a non-empty argument).
`count`	• See note on whitespace in the General Information section.

contd./

Function Name	**Notes**

`current-date,`
`current-dateTime,`
`current-time`
	• The current date and time is taken from the system clock.
	• The timezone is taken from the implicit timezone provided by the evaluation context; the implicit timezone is taken from the system clock.
	• The timezone is always specified in the result.

`deep-equal`	• See note on whitespace in the General Information section.
`doc`	• An error is raised only if no XML file is available at the specified location or if the file is not well-formed. The file is validated if a schema is available. If the file is not valid, the invalid file is loaded without schema information.
`id`	• In a well-formed but invalid document that contains two or more elements having the same ID value, the first element in document order is returned.
`in-scope-prefixes`	• Only default namespaces may be undeclared in the XML document. However, even when a default namespace is undeclared on an element node, the prefix for the default namespace, which is the zero-length string, is returned for that node.
`last`	• See note on whitespace in the General Information section.
`lower-case`	• The ASCII character set only is supported.

`normalize-unicode`	• Not supported.
`position`	• See note on whitespace in the General Information section.

contd./

Function Name	**Notes**
`resolve-uri`	• If the second, optional argument is omitted, the URI to be resolved (the first argument) is resolved against the base URI from the static context, which is the URI of the XSLT stylesheet or the base URI given in the prolog of the XQuery document. • The relative URI (the first argument) is appended after the last `"/"` in the path notation of the base URI notation. • If the value of the first argument is the zero-length string, the base URI from the static context is returned, and this URI includes the file name of the document from which the base URI of the static context is derived (e.g. the XSLT or XML file).
`static-base-uri`	• The base URI from the static context is the base URI of the XSLT stylesheet or the base URI specified in the prolog of the XQuery document. • When using XPath Evaluator in the XMLSpy IDE, the base URI from the static context is the URI of the active XML document.
`upper-case`	• The ASCII character set only is supported.

25.2 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- OS and Memory Requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode Support
- Internet Usage

25.2.1 OS and Memory Requirements

Operating System
This software application is a 32-bit Windows application that runs on Windows NT 4.0, Windows 2000, and Windows XP.

Memory
Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. The memory requirement increases exponentially with the size of the document. For example, a 512kB document would typically require about 2MB of RAM, whereas a 5MB document can consume up to 50MB. Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, memory can rapidly be depleted.

25.2.2 Altova XML Parser

When opening any XML document, the application uses its built-in validating parser (the Altova XML Parser) to check for well-formedness, validate the document against a schema (if specified), and build trees and Infosets. The Altova XML Parser is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in Altova XML Parser implements the Final Recommendation of the W3C's XML Schema specification. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the Altova Parser, so that Altova products give you a state-of-the-art development environment.

25.2.3 Altova XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery 1.0 Engines. Documentation about implementation-specific behavior for each engine is in the section Engine Information, in Appendix 1 of the product documentation, should that engine be used in the product.

These three engines are also available in the AltovaXML package, which can be downloaded from the Altova website free of charge. Documentation for using the engines is available with the AltovaXML package.

25.2.4 Unicode Support

Unicode is the new 16-bit character-set standard defined by the Unicode Consortium that provides a unique number for every character,

- no matter what the platform,
- no matter what the program,
- no matter what the language.

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters: for example, the European Union alone requires several different encodings to cover all its languages. Even for a single language like English, no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

These encoding systems used to conflict with one another. That is, two encodings used the same number for two different characters, or different numbers for the same character. Any given computer (especially servers) needs to support many different encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

Unicode is changing all that!
Unicode provides a unique number for every character, no matter what the platform, no matter what the program, and no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Base and many others.

Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends.

Incorporating Unicode into client-server or multi-tiered applications and web sites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single web site to be targeted across multiple platforms, languages and countries without re-engineering. It allows data to be transported through many different systems without corruption.

Windows NT4.0/2000/XP

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Windows NT typically includes support for all common single-byte writing-systems in its Arial, Times, and Courier New fonts and will additionally include all required fonts for the writing-system in your own country (i.e. if you install the Japanese version of Windows NT you will automatically have fonts that support the Katakana, Hiragana, and Kanji writing-systems as well as the input-methods and dictionaries to enter Kanji and to switch between Katakana and Hiragana). If you wish to edit any document from a foreign writing-system, you may want to install additional Windows NT components for that writing-system or purchase special Unicode fonts for these writing-systems (such fonts are available from all leading type vendors).

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. Consequently you may encounter XML documents that contain "unprintable" characters, because the font you have selected does not contain the required glyphs. Therefore it can sometimes be very useful to have a font that covers the entire Unicode range - especially when editing XML documents from all over the world.

The most universal font we have encountered is a typeface called Arial Unicode MS that has been created by Agfa Monotype for Microsoft. This font contains over 50,000 glyphs and covers the entire set of characters specified by the Unicode 2.1 standard. It needs 23MB and is included with Microsoft Office 2000.

We highly recommend that you install this font on your system and use it with the application if you are often editing documents in different writing systems. This font is not installed with the "Typical" setting of the Microsoft Office setup program, but you can choose the Custom Setup option to install this font.

In the `/Examples` folder in your application folder you will also find a new XHTML file called `Unicode-UTF8.html` that contains the sentence "When the world wants to talk, it speaks Unicode" in many different languages ("Wenn die Welt miteinander spricht, spricht sie Unicode") and writing-systems (世界的に話すなら、Unicode です。) - this line has been adopted from the 10th Unicode conference in 1997 and is a beautiful illustration of the importance of Unicode for the XML standard. Opening this file will give you a quick impression on what is possible with Unicode and what writing systems are supported by the fonts available on your PC installation.

Right-to-Left Writing Systems

Please note that even under Windows NT 4.0 any text from a right-to-left writing-system (such as Hebrew or Arabic) is not rendered correctly except in those countries that actually use right-to-left writing-systems. This is due to the fact that only the Hebrew and Arabic versions of Windows NT contains support for rendering and editing right-to-left text on the operating system layer.

25.2.5 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (**Help | Registration**), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- If you use the Open URL... dialog box to open a document directly from a URL (**File | Open URL**), that document is retrieved through a http (port 80) connection. (*This functionality is available in XMLSpy and Authentic Desktop.*)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, it is also retrieved through a http (port 80) connection once you validate the XML document. This may also happen automatically upon opening a document if you have instructed the application to automatically validate files upon opening in the File tab of the Options dialog (**Tools | Options**). (*This functionality is available in XMLSpy and Authentic Desktop.*)
- If you are using the Send by Mail... command (**File | Send by Mail**) in XMLSpy, the current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.

Note: All Internet communication is initiated only when directly requested by you. from you! (This functionality is important in an XML development environment since XML is, after all, a technology closely related to the Internet.)

25.3 License Information

This section contains:

- Information about the distribution of this software product
- Information about the copyrights related to this software product
- The End User License Agreement governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

25.3.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge before making a purchasing decision.
- Once you decide to buy the software, you can place your order online at the Altova website and immediately get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period
After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into this evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you have to purchase an End User License Agreement, which is delivered in the form of a key-code that you enter into the Registration dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

Distributing the product
If you wish to share the product with others, please make sure that you distribute only the installation program, which is a convenient package that will install the application together with all sample files and the onscreen help. Any person that receives the product from you is also automatically entitled to a 30-day evaluation period. After the expiration of this period, any other user must also purchase a license in order to be able to continue using the product.

For further details, please refer to the End User License Agreement at the end of this section.

25.3.2 License Metering

Your Altova product has a built-in license metering module that helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license
When the application starts up, it sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application. Other than that, it will not attempt to communicate over a network. If you are not connected to a LAN, or are using dial-up connections to connect to the Internet, the application will not generate any network traffic at all.

Multi license
If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to ensure that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses..

Please note that your Altova product at no time attempts to send any information out of your LAN or over the Internet. We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (*see* http://www.isi.edu/in-notes/iana/assignments/port-numbers *for details*) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.

25.3.3 Copyright

All title and copyrights in this software product (including but not limited to images, photographs, animations, video, audio, music, text, and applets incorporated in the product), in the accompanying printed materials, and in any copies of these printed materials are owned by Altova GmbH or the respective supplier. This software product is protected by copyright laws and international treaty provisions.

- This software product ©1998-2006 Altova GmbH. All rights reserved.
- The Sentry Spelling-Checker Engine © 2000 Wintertree Software Inc.
- STLport © 1999, 2000 Boris Fomitchev, © 1994 Hewlett-Packard Company, © 1996, 1997 Silicon Graphics Computer Systems, Inc, © 1997 Moscow Center for SPARC Technology.
- Scintilla © 1998–2002 Neil Hodgson <neilh@scintilla.org>.
- "ANTLR Copyright © 1989-2005 by Terence Parr (www.antlr.org)"

All other names or trademarks are the property of their respective owners.

25.3.4 Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensor:

Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Software License Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Software License Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the Software in your possession or control. Please go to our Web site at http://www.altova.com/eula to download and print a copy of this Software License Agreement for your files and http://www.altova.com/privacy to review the privacy policy.

1. SOFTWARE LICENSE

(a) **License Grant.** Upon your acceptance of this Software License Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license to install and use a copy of the Software on your compatible computer, up to the Permitted Number of computers. The Permitted Number of computers shall be delineated at such time as you elect to purchase the Software. During the evaluation period, hereinafter defined, only a single user may install and use the software on one computer. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Software License Agreement governs your use of all of the software included in the Suite. If you have licensed SchemaAgent, then the terms and conditions of this Software License Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the purposes described in the accompanying documentation. In addition, if you have licensed XMLSpy Enterprise Edition or MapForce Enterprise Edition, or UModel, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile into executable form the complete generated code comprised of the combination of the Restricted Source Code and the Unrestricted Source Code, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer to a third party the Restricted Source Code, unless said third party already has a license to the Restricted Source Code through their separate license agreement with Altova or other agreement with

Altova. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise permitted in Section 1(h) reverse engineering of the Software is strictly prohibited as further detailed therein.

(b) **Server Use.** You may install one copy of the Software on your computer file server for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers. If you have licensed .SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) **Concurrent Use**. If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time. The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses.

(d) **Backup and Archival Copies.** You may make one backup and one archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(e) **Home Use.** You, as the primary user of the computer on which the Software is installed, may also install the Software on one of your home computers for your use. However, the Software may not be used on your home computer at the same time as the Software is being used on the primary computer.

(f) **Key Codes, Upgrades and Updates.** Prior to your purchase and as part of the registration for the thirty (30) -day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the update replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or update in addition to the Software that it is replacing. You agree that use of the upgrade of update terminates your license to use the Software or portion thereof replaced.

(g) **Title.** Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Software License Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Software License Agreement, are your property.

(h) **Reverse Engineering.** Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(i) **Other Restrictions.** You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Software License Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Software License Agreement. You will comply with applicable law and Altova's instructions regarding the use of the Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Software License Agreement and to ensure their compliance with these restrictions. you agree that you are solely responsible for the accuracy and adequacy of the software for your intended use and you will indemnify and HOLD harmless ALTOVA from any 3rd party suit to the extent based upon the accuracy and adequacy of the software in your use. without limitation, The Software is not intended for use in the operation of nuclear facilities, aircraft navigation, communication systems or air traffic control equipment, where the failure of the Software could lead to death, personal injury or severe physical or environmental damage.

2. INTELLECTUAL PROPERTY RIGHTS

Acknowledgement of Altova's Rights. You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that trademark. XMLSpy, Authentic, StyleVision, MapForce, Markup Your Mind, Axad, Nanonull, and Altova are trademarks of Altova GmbH (registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows 95, Windows 98, Windows NT, Windows 2000 and Windows XP are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Software License Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer each of this Software License Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Software License Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-release Software"), then this Section applies. In addition, this section applies to all evaluation and/or demonstration copies of Altova software ("Evaluation Software") and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Software License Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU **"AS-IS" WITH NO WARRANTIES FOR USE OR PERFORMANCE,** AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD $50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Software License Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) **Limited Warranty and Customer Remedies.** Altova warrants to the person or entity that first purchases a license for use of the Software pursuant to the terms of this Software License Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by Altova shall be substantially as described in Section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. Altova's and its suppliers' entire liability and your exclusive remedy shall be, at Altova's option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet Altova's Limited Warranty and which is returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or any other malicious external code. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does not apply to Evaluation and/or Pre-release Software.

(b) **No Other Warranties and Disclaimer.** THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA OR ITS SUPPLIER'S BREACH OF WARRANTY. ALTOVA AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ALTOVA AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) **Limitation Of Liability.** TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTOVA OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ALTOVA'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS SOFTWARE LICENSE AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, Altova's liability shall be limited to the greatest extent permitted by law and the limitations or exclusions of warranties and liability contained herein do not prejudice applicable statutory consumer rights of person acquiring goods otherwise than in the course of business. The disclaimer and limited liability above are fundamental to this Software License Agreement between Altova and you.

(d) **Infringement Claims.** Altova will indemnify and hold you harmless and will defend or settle any claim, suit or proceeding brought against you by a third party that is based upon a claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost with such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova's legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova's legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Software License Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA'S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT. This indemnity does not apply to infringements that would not be such, except for customer-supplied elements.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional "Support & Maintenance Package(s)" ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30)-day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, **you are eligible to receive the version of the Software edition** that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP's Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only, and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova's business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or update, which is made available at certain times at Altova's sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in

maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.
Updating Software may require the updating of software not covered by this Software License Agreement before installation. Updates of the operating system and application software not specifically covered by this Software License Agreement are your responsibility and will not be provided by Altova under this Software License Agreement. Altova's obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Software License Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova's written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova's specified release level; or (v) use of the Software with other software without Altova's prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) **License Metering**. Altova has a built-in license metering module that helps you to avoid any unintentional violation of this Software License Agreement. Altova may use your internal network for license metering between installed versions of the Software.

(b) **Software Activation**. **Altova's Software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the Software and to improve customer service. Activation is based on the exchange of license related data between your computer and the Altova license server. You agree that Altova may use these measures and you agree to follow any applicable requirements.**

(c) **LiveUpdate**. Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(d) **Use of Data.** The terms and conditions of the Privacy Policy are set out in full at http://www.altova.com/privacy and are incorporated by reference into this Software License Agreement. By your acceptance of the terms of this Software License Agreement or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Software License Agreement and/or the Privacy Policy as revised from time to time. European users understand and consent to the processing of personal information in the United States for the purposes described herein. Altova has the right in its sole discretion to amend this provision of the Software License Agreement and/or Privacy Policy at any time. You are encouraged to review the terms of the Privacy Policy as posted on the Altova Web site from time to time.

8. TERM AND TERMINATION

This Software License Agreement may be terminated (a) by your giving Altova written notice of termination; or (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Software License Agreement and fail to cure such breach within ten (10) days after notice from Altova. In addition the Software License Agreement governing your use of a previous version that you have upgraded or updated of the Software is terminated upon your acceptance of the terms and conditions of the Software License Agreement accompanying such upgrade or update. Upon any termination of the Software License Agreement, you must cease all use of the Software that it governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(g), (h), (i), 2, 5(b), (c), 9, and 10 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Software License Agreement. Manufacturer is Altova GmbH, Rudolfsplatz, 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. GENERAL PROVISIONS

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Software License Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Software License Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Software License Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Software License Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

This Software License Agreement contains the entire agreement and understanding of the parties with respect to the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Software License Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova's Web site for Altova and the address shown in Altova's records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Software License Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party's right at a later time to enforce the same or any other term of this Software License Agreement. This Software License Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or threatened breach of this Software License Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Software License Agreement. If, for any reason, any provision of this Software License Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Software License Agreement, and this Software License Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2005-05-05

Index